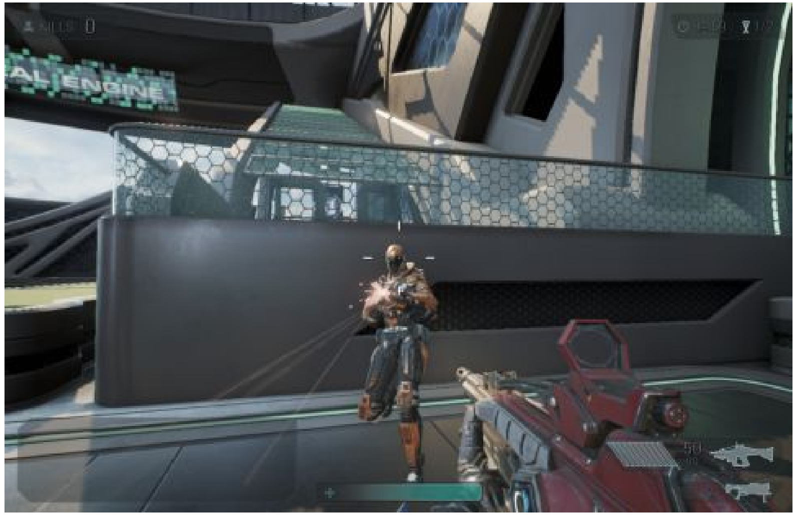
Modelling Human-like Behavior through Reward-based Approach in a First-Person Shooter Game



Research Head: Ilya Makarov

Project participants: Peter Zyuzin, Pavel Polyakov,Ivan Guschenko-Cheverda, Maxim Uriev, et. all.

Content

- First-Person Shooter
- Game AI

- Weapon Selection
- Path Planning
- Path Finding

First-Person Shooter Video Game

A first-person shooter game is a special genre of video games simulating combat actions with projectile-based weapons through an eyes perspective of human-like model placed in virtual world.

At the round start each player is assigned to one of competing teams and all the players of one team are spawned at their team base location with starting weapon combination and full health.

In what follows player can move and use weapon in order to survive or kill enemies.

AI in FPS Games

Computer models addressing intelligence tests have different purposes and applications:

- to advance AI by the use of challenging problems,
- to use them for the evaluation of AI systems,
- to better understand measures of intelligence,
- to better understand what human intelligence is.

The main criterion to verify the quality of a game AI is the level of compliance for non-playable characters’ (NPCs) actions to distinguish computer-controlled and human players by human judges through Alan Turing test for game BOTs.

Weapon Selection using FALCON

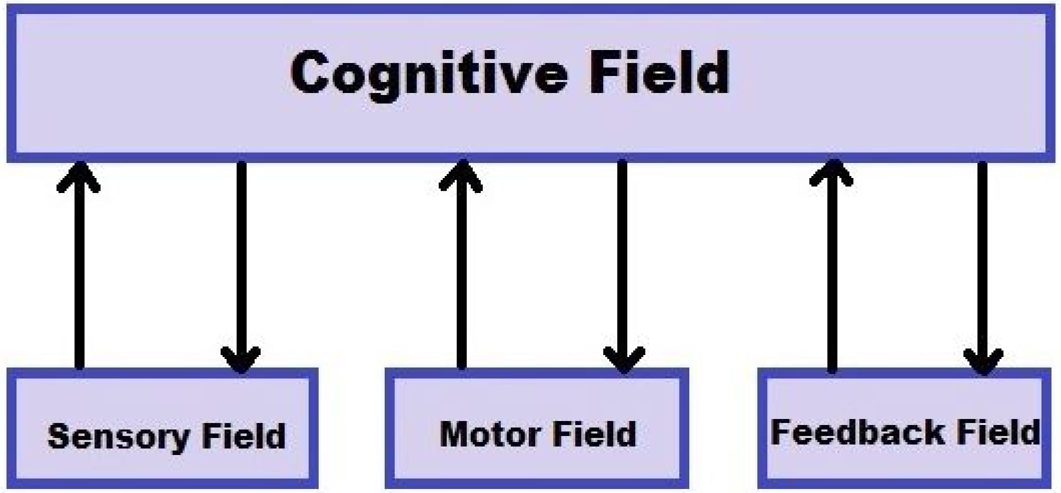


Figure 1. FALCON Architecture

FALCON –
Fusion
Architecture for
Learning,
COgnition and
Navigation.

Wang, D., Subagdja, B., Tan, A., Ng, G.: Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games. In: Proc. IAAAI 2009 (2009)

Weapon selection with FALCON

- **States:**

– normalized distance between the BOT and its enemy;

– absolute value of projection of enemy’s velocity vector onto plane that is normal to BOT's aiming vector.

- **Actions - use:** machine-gun or shotgun or sniper rifle.
- **Reward:** $r = (a + b * distance) * damage$ (we find optimal $a = 1$, $b = 9$)

$distance$ - normalized distance between the BOT and the enemy;

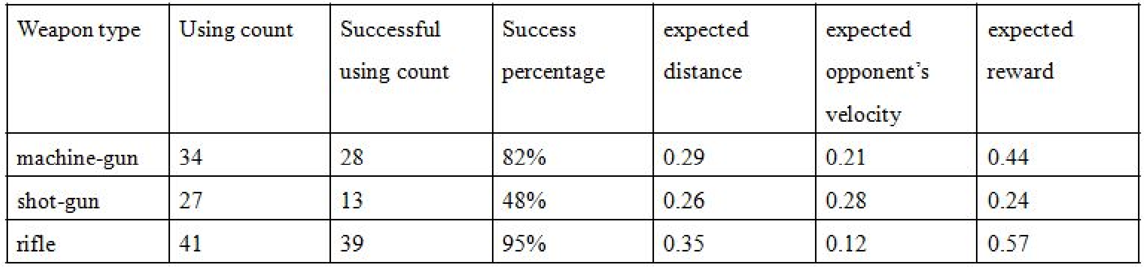
$damage$ - normalized value of damage that the BOT inflicts to the enemy.

Modified FALCON

- We delete a neuron if count of successful usages of this neuron exceeds count of unsuccessful usages of the neuron;
- We delete a neuron if its activity lead to receiving zero reward several times in a row;
- We restrict the size of the cognitive field. If it is overcrowded with neurons we delete a half of neurons with minimal value of expected reward.

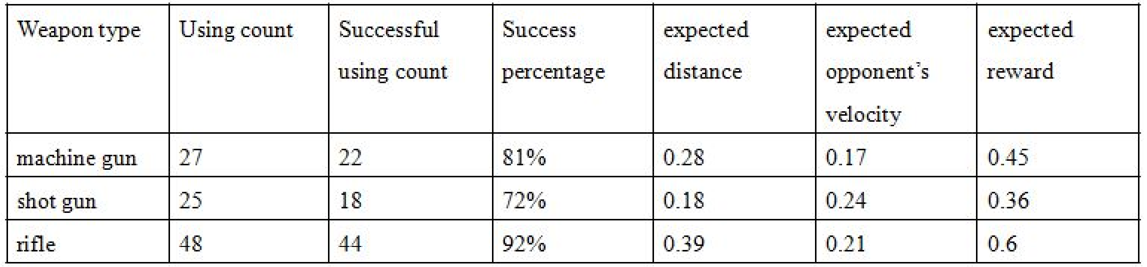
FALCON Experiment Results

Initial algorithm



Weapon type	Using count	Successful using count	Success percentage	expected distance	expected opponent's velocity	expected reward
machine-gun	34	28	82%	0.29	0.21	0.44
shot-gun	27	13	48%	0.26	0.28	0.24
rifle	41	39	95%	0.35	0.12	0.57

Modificated algorithm



Weapon type	Using count	Successful using count	Success percentage	expected distance	expected opponent's velocity	expected reward
machine gun	27	22	81%	0.28	0.17	0.45
shot gun	25	18	72%	0.18	0.24	0.36
rifle	48	44	92%	0.39	0.21	0.6

Table 1, 2. Results of FALCON and Modified FALCON Learning

Path Planning

Path planning and path nding problems play one of the main topics in robotic and automation elds, especially for dynamically changing environments.

Voronoi diagrams are the simplest case of a k-nearest neighbors classification rule for points $\{p_1, \ldots, p_n\}$ with k=1.

$$VD(p_i) = \{x : |p_i - x| \leq |p_j - x|, \forall j \neq i, x \in \mathbb{R}^2\}$$

In game programming, Voronoi diagrams are used to make a partition of a navigation mesh to find a collision free path in both global and local environments.

The path is a piecewise linear path or a curve smoothed with the help of

- Splines through way points on map,
- Composite Bezier curves based on avoiding obstacles.

Map Tactical Properties and Penalties for Smoothing

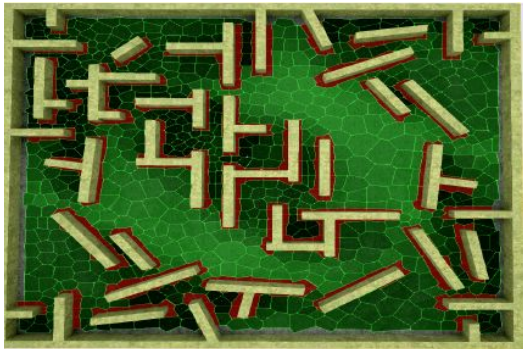


Figure 3. Tactical NavMesh.

Navigation consists of the following steps:

1. BOT makes a query to navigation system;
2. Navigation system uses I-ARA* algorithm finding a sequence of adjacent polygons on navigation mesh;
3. A sequence of polygons is converted into a sequence of randomly chosen points on a common edge;
4. BOT receives a sequence of points and build a collision free path to walk.

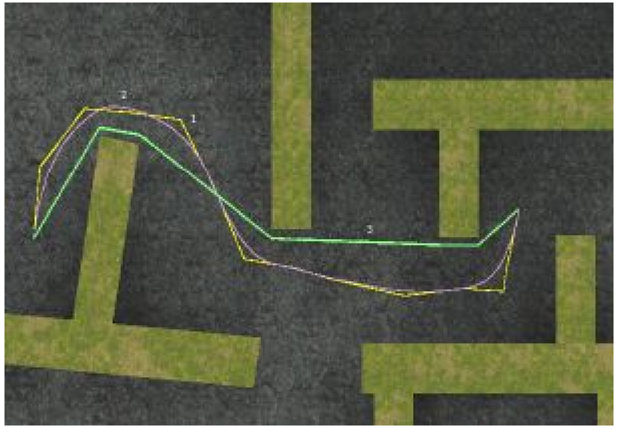
General Penalties:
BaseCost();
BaseEnterCost();
NoPathFlag();
Visibility();

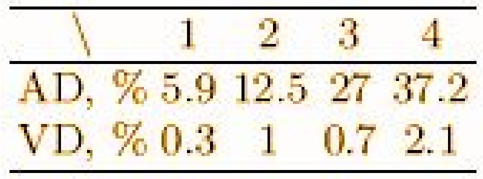
List of Penalties for Path Planning:
GetPenaltyForRotation();
GetPenaltyMultiplierForCrouch();
GetPenaltyForJump();
GetAdditionalPenalty(PreviousPolygon,NextPolygon);

Experiments on Path Planning

We take 1000 different start and end locations on the map shown on Figure 4. The average difference (AD, %) and variance of difference (VD, %) from the shortest path length for the next paths were calculated during the experiment:

1. Piecewise path with visibility penalty set to 0;
2. Piecewise path with visibility penalty set to10;
3. Smoothed path with visibility penalty set to 0;
4. Smoothed path with visibility penalty set to 10.





\	1	2	3	4
AD, %	5.9	12.5	27	37.2
VD, %	0.3	1	0.7	2.1

Table 1. Comparison with the Shortest Path Length

I-ARA* Search Algorithm

Moving-target search, where a hunter has to catch a moving target, is an important problem for video game developers.

A* path planning algorithm cannot always guarantee the continuity of a player’s movements when the allocated time is limited.

Anytime Repairing A*(*ARA)** can get a sub-optimal solution quickly, and then work on improving the solution until the allocated time expires.

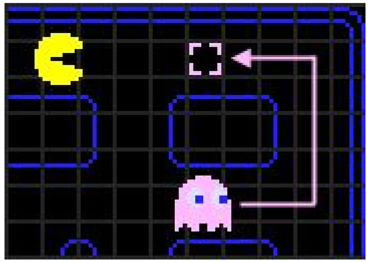
$f(s)=g(s)+eps\times h(s, t)$, s - path, $g(s)$ - current shortest path length from start, $h(s,t)$ - estimated cost of reaching target t from s, eps - weight of repair iteration

Incremental ARA* decreases search time even without anytime condition compared to repeated A* .

Decreasing Time for Long Distances Path Planning

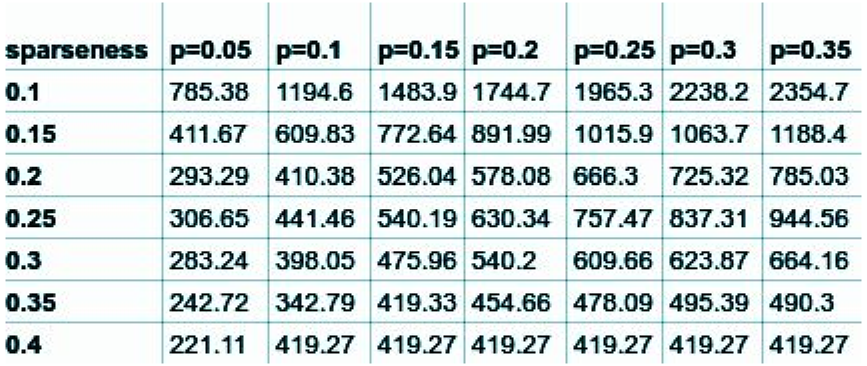
The main idea is to **walk a certain percentage of suboptimal** path without calculations of repeated ARA*.

- Positive effects: significant decreasing of time for calculations. Difference in paths' length is less than 10% for dense mazes.
- Negative effects: new iterations have longer durations for some positions. Lack of anytime property in a new algorithm.



I-ARA* Comparison

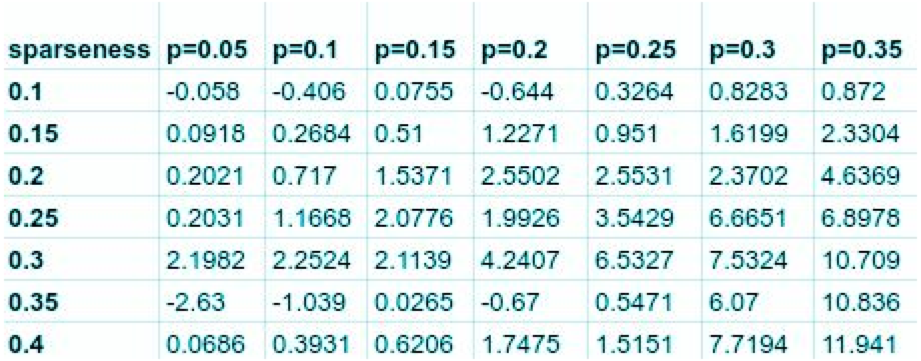
We examined original I-ARA* and our algorithm for mazes of sizes 300x300 and 600x600 with different sparseness and percentage (p), and compared overall search times and path lengths.



sparseness	p=0.05	p=0.1	p=0.15	p=0.2	p=0.25	p=0.3	p=0.35
0.1	785.38	1194.6	1483.9	1744.7	1965.3	2238.2	2354.7
0.15	411.67	609.83	772.64	891.99	1015.9	1063.7	1188.4
0.2	293.29	410.38	526.04	578.08	666.3	725.32	785.03
0.25	306.65	441.46	540.19	630.34	757.47	837.31	944.56
0.3	283.24	398.05	475.96	540.2	609.66	623.87	664.16
0.35	242.72	342.79	419.33	454.66	478.09	495.39	490.3
0.4	221.11	419.27	419.27	419.27	419.27	419.27	419.27

Table 3. Time decreasing, %

I-ARA* Comparison



sparseness	p=0.05	p=0.1	p=0.15	p=0.2	p=0.25	p=0.3	p=0.35
0.1	-0.058	-0.406	0.0755	-0.644	0.3264	0.8283	0.872
0.15	0.0918	0.2684	0.51	1.2271	0.951	1.6199	2.3304
0.2	0.2021	0.717	1.5371	2.5502	2.5531	2.3702	4.6369
0.25	0.2031	1.1668	2.0776	1.9926	3.5429	6.6651	6.8978
0.3	2.1982	2.2524	2.1139	4.2407	6.5327	7.5324	10.709
0.35	-2.63	-1.039	0.0265	-0.67	0.5471	6.07	10.836
0.4	0.0686	0.3931	0.6206	1.7475	1.5151	7.7194	11.941

Table 4. Path length increasing, %

Thank you!

E-mail: iamakarov@hse.ru