Computational Intelligence in Games

Julian Blank, Frederick Sander

Otto-von-Guericke-University Magdeburg, Germany
julian.blank@st.ovgu.de
frederick.sander@st.ovgu.de

Abstract. Summarize the paper in a paragraph of two. It should contain at least 70 and at most 150 words. You should motivate the research done, give some details about the experiments run and briefly mention the most important findings.

1 Introduction

Give an introduction to the problem tackled. Why is it important? Outline the rest of the document.

2 Literature Review

Perform a literature review: What has been done before in this field? What is the main technique/s used in the paper, and what has it/they been used for in the literature before? Give references to the most relevant work published. Example: [2].

3 Background

There are several basic approaches to write an artificial intelligence (AI) for a game. In this project we were forced to investigate different ideas at three different research areas: heuristic based search, reinforcement learning and nature inspired algorithm. Of course there is an overlap between these approach. All of them try to find the next best step for the agent by iterating through a search tree. This is build in base of all possible game steps that could done by the agent. One naive approach of iterating through the whole search tree is not possible, because there is a time limitation. Generally this leads to solve the trade-off between iterating similar to a breadth-first or depth-first search. The heuristic approach forces the second by using a estimation function for looking many steps ahead.

3.1 Heuristic Based Search

A heuristic is used to evaluate a game state by putting several facts into one number. When we have to decide which current active branch of a search tree should be iterated this score might help us. One common idea to estimate the distance to the target is the manhatten distance [1].

In a two dimensional space the manhatten distance is calculated by

$$dist(u, v) = |x_1 - x_2| + |y_1 - y_2| \tag{1}$$

adding the absolute value of the difference for the x and the y axis. The input consists always of the points that have one value for each dimension. This could be extended for a n-dimensional space as well. When thinking of a way at a grid this is always a path with one rectangle waypoint 1.

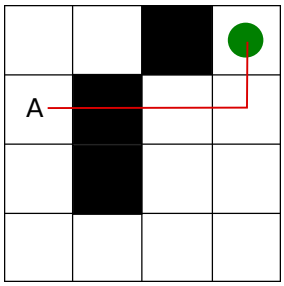

Fig. 1: Manhatten distance

3.1.1 Greedy Greedy-algorithms are a whole class of algorithms and strategies. All of them follow a specific scheme/rule. They are iterative and choose in every step the state with the best reward. The state is in most cases a node which represents the state of the algorithm. The advantage of greedy algorithms is that they are very fast but on the other hand they are not optimal they often only find a local optima and not the global one. The advantage and disadvantage is caused by the greedy approach.

3.1.2 One Step Lookahead One step lookahead is a very simple tree search algorithm which follows the greedy approach. The actual state is the root node. From this node we only look one step ahead to all nodes which are connected by one edge and compute a heuristic value or another kind of reward value for these nodes.

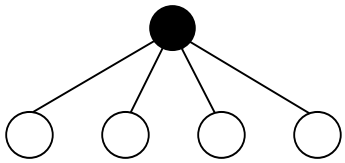

Fig. 2: Search tree for One Step Lookahead

After that the algorithm terminates and we pick the node with the best value.

3.1.3 AStar The A* tree search algorithm is a modification of the dijkstra algorithm and also belongs to the class of greedy algorithms. The Algorithm finds the shortest path between two nodes. In difference to normal greedy algorithms A* is a optimal algorithm, it finds a solution when a solution exists (in this case the shortest path). The algorithm uses a heuristic to estimate the shortest path. The value f(x) of a node N is the sum of its heuristic value h(x) and the costs from the start-node to N g(x).

$$f(x) = h(x) + g(x) \tag{2}$$

A* contains two sets of nodes, the openlist and the closedlist. In every step of the algorithm the Node N with the lowest f(x) value in the openlist is put on the closedlist and all its connected nodes, which are not in the closedlist, are put in the openlist (with reference to their fahter N). If a connected node is already in the closedlist but the new generated value f'(x) is lower than f(x) then f(x) will be replaced by f'(x) and the new father-reference is N. The openlist contains all the nodes to which the path is already known and which can be checked in the next step, the closedlist contains every visited and checked node. When the actual node is the goal-node, the algorithm terminates. To generate the path, the algorithm goes back from the goal-node to the start node (guided by the father-references).

3.2 Reinforcement learning

Reinforcement learning, below RL, is a field in Machine learning which is a section of Artificial Intelligence. RL methods are mostly used by agents in an environment called Markov decision process, below MDP. MDP is a mathematical description of decision processes. They have different states S and some actions A which are available in the actual state. Every timestep the agent chooses an action a and the process switches from state s_a to s_n. The probability to go over form a state S to another state S' by any action A can be described as

$$G : S * S * A \to [0, 1]$$

and the reward given to the agent can be described by this:

$$R : S * S * A \to \mathbb{R}$$

So that

$$(s_a, s_n, a) \to p(s_n | s_a, a)$$

would describe the probability p to go over in the state s_n, given the actual state s_a and the choosen action a and

$$(s_a, s_n, a) \to r$$

shows its corresponding reward.

In differ to other learn methods and approaches like the (semi)supervised learning, RL algorithms never use information which they do not figured out themselves, so no correct samples were given to the algorithm. The only information is the reward given to the agent and some additional information like heuristic values, depending of the specific algorithm. A big problem problem in RL is the conflict between exploration of new and unvisited areas of the solution room and exploitation which is the improvement of already found solutions. ...

3.2.1 Monte Carlo Tree Search

Monte Carlo Tree Search, below MCTS, is a class of RL algorithms. It is the most important concept in this paper. MCTS needs a tree of nodes which represent the different states, the edges represent the actions used by the agent to get to this node. The MCTS algorithm traverses to this tree and expands it. To find the global optimum a good balancing ratio between exploration and exploitation is required.

[maybe picture from his paper and pseudocode]

The general MCTS algorithm has four steps, selection, expansion, simulation and backpropagation. In the selection the algorithm starts at freddy working on it

3.2.2 Temporal difference methods

freddy working on it

3.2.3 Q-learning

freddy working on it

3.3 Nature inspired

The nature is solving problems by applying different approaches instinctively.

3.3.1 Neural nets

Our brain solves many problems that could not be solved by algorithms yet. Many researcher tries to explore the process of the human brain. Since there are a lot of results by now the neural nets were born that do in principle the same as neurons.

3.3.2 Evolutionary algorithm

3.3.3 pheromones

4 Techniques Implemented

Describe the controllers you created. Explain how they work, in which techniques are they based on (probably from previous section), and include figures and pseudocode as needed.

4.1 Stay Alive

pessimistic iteration

grid search

4.2 Heuristic based

4.3 MCTS Tree

- act choosig -¿ most visited node
- tree policy
- default policy

4.4 Evolutionary algorithm

- basics mutation score crossover
- sliding window
- adaptive path length
- heuristic switch

4.5 Game Detection

Another technique we have implemented is a detection of a known game. We know the 10 games from the test-set and the 10 games from the validation-set, the 10 test-set games are unknown. The Goal of the Gamedetection is to improve the score and the number of wins in the two known game-sets without decreasing the score and the number of wins in the unknown test-set. To do that, the standard parameters of the Algorithm are used when no game is detected, otherwise the optimal parameters for this game are used. These parameters we figured out for some difficult games in which the standard parameters give bad results. To detect a game we generate a String of all Objects (npc, movable, immovable, ect...) and store the Hash value of this String. All the hashes from the known games are stored, in a running Game we generate another String of these Objects and compare the generated hash value to the stored ones.

5 Experimental Study

Detail the experimental setup used to test the different algorithms. Present the results in an understandable manner (graphics, tables, etc.). Draw conclusions about what things worked (and why) and which didn't (and why).

5.1 Among each Approach

5.2 General

6 Conclusions and Future Work

Explain the main contributions of this work: what are the most important findings. Finally, explain how could this work be extended. What would be the next steps?

References

1. P. Barrett, “Euclidean distance: raw, normalized, and doublescaled coefficients,” September 2005, pbarrett.net/ [Online; posted 7-January-2015].
2. C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods.” *IEEE Trans. Comput. Intellig. and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/tciaig/tciaig4.html#BrownePWLCRTPSC12