Artificial Intelligence for Games

4th Project

Group 29:

Ricardo Silva, student 78414

Tomasz Topczewski, student 85986

Abstract

In this project, we created an AI to play our own version of the popular puzzle game Tetris. Besides from using MCTS with biased playout from project3 we also added the parameter learning algorithms Hill Climb, Simulated Annealing and Genetic Algorithms and included them in Fast Evolutionary MCTS hoping to get the best weights for the biased playout heuristic. We aimed to compare the variants of parameter learning on effectiveness levels. The results obtained are presented below.

Tetris Implementation

Because we were not able to find any implementation in C# that suited our needs we implemented our own simple version of Tetris which aims to provide all the needed functionality to simulate the game in a way that is playable by an AI. The structure is close to the one used in project 3 where we have the State class, Action class, Board class and Pieces class.

We made it so that only the first 3 pieces are known to the MCTS search and every update another random piece is added to simulate the behaviour of a normal Tetris game where the player can see the current piece and some of the following pieces.

To draw the board of the current state we coded a Drawer class which takes a board and draws cubes in the occupied spaces of the board.

Biased Playout

In this project, we needed some way to evaluate the quality of each state so we implemented 5 simple heuristics. With those 5 heuristics, we created a master heuristic that computes a weighted sum of the first 5 heuristics. The weights of this sum were our target for the learning process.

Unfortunately, we were not able to use the actions to calculate this value like in project 3 because in this case the actions don’t present enough information to conclude anything. We needed to compute the application of that action to a copy of the current state to get the heuristic value which has a worse performance.

The heuristics implemented here the following:

- Bumpiness – Tries to get some measure of how rough the board is by computing the difference of heights between every pair of columns;
- FreeSpaces – Computes the number of empty spaces in the board;
- HighestBlock – Return the height of the highest column;
- Score – Return the score of the current state;
- AverageHeight – Returns the average height of all columns.

Parameter Learning

To reach optimal parameters, we used Fast Evolutionary Monte Carlo Tree Search to learn the weights of each heuristic in the sum. This algorithm takes advantage on Monte Carlo Tree Search algorithm with improvements like biased playout to reach the optimal parameters to run the program. To reach this goal we used three different learning methods. The easiest one we chose just to have some comparison was Hill Climb method, which in our implementation simply changes value of one of the parameters and checks if new fitness function value is better then it's predecessor. The second implemented algorithm was Simulated Annealing, which is similar to the Hill Climb and the most important change is introducing the probability of choosing the worse solution which changes with the development of algorithm being represented by value called temperature. Third option is genetic algorithm, which introduces more complexity and even more randomness. It starts it's work by generating group of random solutions similar to the initial one and evolves them by selection and mutation functions in order to keep the best solutions but maintain variety as long as possible.

Comparison of the parameter learning algorithms:

The tests were done using 20 random pieces equal for every run and algorithm.

We did 10 simulations for each set of weights and computed the average score.

	Run Time = 5 min	**Run Time = 10 min**	**Run Time = 15 min**
Hill Climbing	BestAverageScore: 300 Weights: {3.88, 5.74, 4.74, -1.66, 5.31}	BestAverageScore: 500 Weights: {-2.08, 2.22, 1.53, 4.50, 6.05}	BestAverageScore: 400 Weights: {0.80, 2.05, -0.15, -0.07, 0.14}
Simulated Annealing	BestAverageScore: 400 Weights: {-143.11, -478.07, 623.28, -301.39, 80.91}	BestAverageScore: 400 Weights: {-377.12, 224.30, 50.56, -28.60, -41.98}	BestAverageScore: 500 Weights: {-831.0909 -255.1926 -373.157 -44.83672 -774.6692
Genetic Algorithm	BestAverageScore = 400 Weights: 9.066021 -41.67606 2.981536 -3.757051 -11.21428	BestAverageScore = 400 Weights 4.338678 7.909036 46.83519 36.48143 2.151486	BestAverageScore = 500 Weights -4.08502 -20.81908 -52.5731 0.07976406 6.183765

As we can see the different algorithms achieve quite similar results which is kind of surprising to us if we take into account that Hill Climbing is a much simpler algorithm than the other. Maybe if we had used a bigger number of pieces in the tests the results would be different but we didn't had time to further test the algorithms.

MCTS with biased playout with best weights:

This time we have unlimited pieces and let the game run until the AI losses. The AI only sees the first 3 pieces, in each update a new piece is added. To get better results we repeated each test 5 times and computed the average score.

Time Per Action:	1s	2s	3s	4s
Average Score:	100	300	1000+	1500+

Optimizations

We profiled the projected and optimized all Tetris related functions as much as possible. Special attention was given to the heuristics and the State.ApplyAction() and State.GenerateChildState() methods because they were the most time consuming and the most called methods.

Discussion

In this project, we learned more about local search algorithms and parameter optimization which are quite interesting to us, specially the genetic algorithm.

It is also interesting how a not so complex AI created in a couple of days can easily play Tetris better than we can.

Regarding learning algorithms, we can say that while they are quite simple, all three of them are local search algorithms which are deeply related to stochastic functions, so with every run these algorithms can behave slightly different therefore can be hard to study in terms of efficiency and effectiveness.