AI Tetris - A CS3243 Project (Group 21)

Lau Kar Rui (A0155936U), Poh Jie, Matilda (A0178867A)

April 13, 2018

1 Introduction

In this project, we create an utility based Tetris-playing AI Agent with utility weights derived through the *Particle Swarm Optimization* algorithm to maximise the number of cleared rows in a single game. A standard Tetris board of 20 rows by 10 columns is used, along with the standard 7 tetrominos (I, O, T, S, Z, J, and L) that are randomly chosen each turn.

The best move is chosen by the *state* that maximizes the utility function. This is done by the summation of the values of 8 different features. Calculating the summation of all the features is trivial, but deciding the best weights for our features to pick the best move is both complex and crucial for our agent to perform to its highest potential.

2 Utility Function

We defined the utility function as a linear function F with each feature $f_i \in Features$ having an assigned weight w_i, where $i = 1...n$, with n being the number of features. Each f_i derives a real value from a state s. The function is then defined as:

$$F(s) = \sum_{i=1}^{n} w_i f_i(s)$$

3 Features Used

Table 3.1 shows the features f_i used to calculate $w_i \in Weights$ when training with the PSO algorithm described in Section 4.2.

Feature	**Description**
RowsCleared	Number of rows cleared
MaxHeightIncrease	The maximum height increase among all columns
AvgHeightIncrease	The average height increase among all columns
AbsoluteDiff	The absolute height difference between all columns
NumHoles	Number of holes; a hole is defined as an empty cell with a non-empty cell above it

ColumnTransition	The total number of column transitions; a column transition is defined as an empty cell adjacent to a filled cell on the same column and vice versa
RowTransition	The total number of row transitions; a row transition is defined as an empty cell adjacent to a filled cell on the same row and vice versa
WellSum	The total number of empty cells above the columns' top heights which are adjacent to filled cells on both sides

Table 3.1: Features used

4 Implementation

The utility function F is applied for every possible state s_i using the `StateCopy` mechanism described in Section 4.1 (with state meaning every possible orientation and landing column of the current landing piece), and the s_i with $\max \{value(F(s_i)) \mid \forall s_i \in States\}$ is chosen to be the current state.

4.1 StateCopy

In order to correctly apply the features, a new `StateCopy` class was created, extending the original `State` class, serving as a clean starting state to apply our features on in order to derive the feature value.

Extra variables like `currentRowsCleared` and `previousTop` is also added to the `StateCopy` class in order to obtain the information needed by various features such as the *RowsCleared* and the *AverageHeight* features.

Using `StateCopy` also allows us to play moves without affecting the original state of the game to find the move with the highest function value.

4.2 Particle Swarm Optimization (PSO)

The PSO algorithm is inspired by the metaphor of social interaction observed between fishes or birds. The basic variant of PSO is used, with a population (a.k.a. swarm) of candidate solutions (a.k.a. particles), which moves around in the search-space of our linear function F, guided by their own best known position in the search-space as well as the entire swarm's best known position to find the best positions (a.k.a. $w_i \in Weights$) for $F(s)$.

The movement of each particle $\vec{x}_i$ for $i = 1...N$, where N is the number of particles in the swarm, is guided by the formulae

$$\vec{v} \leftarrow \omega\vec{v} + \phi_p r_p(\vec{p} - \vec{x}) + \phi_g r_g(\vec{g} - \vec{x})$$
$$\vec{x} \leftarrow \vec{x} + \vec{v}$$

where $\vec{p}$ is the particle's *personal best position* and $\vec{g}$ is the swarm's *global best position*, ϕ_p and ϕ_g being the *social* and *cognitive coefficient* respectively, emulating the swarm's willingness to move in the search-space.

ϕ_p acts as the particle's "memory", causing it to return to its individual best regions of the search space, while ϕ_g guides the particle to move to the localized search-space where the global best position was discovered. r_p and r_g are two random real values $\in [0..1]$ generated every assignment to $\vec{v}$.

ω refers to the *inertia* of the particle, and keeps the particle moving in the same direction it was originally heading. A lower value speeds up convergence of the swarm in the search space, and a higher value encourages exploring the search-space.

Values for these parameters were first chosen based on values discovered by *Van Den Bergh and Engelbrecht* [1] and later optimized with the Meta Optimization algorithm discussed in Section 4.3.

4.3 Meta Optimization of PSO using PSO

Talk about the PSO optimization here

5 Scaling The Algorithm For Big Data

Talk about parallelizing the application here Talk about using Compute Cluster

	Time Taken for 20 iterations (in seconds)
No multithreading	
Multithreading particles	
Multithreading games	

Table 5.1: Time comparison

6 Training Results

After 1000 iterations of our training algorithm described in Section 4.2, we arrived at the weights:

$$\begin{pmatrix} MaxHeightIncrease \\ RowsCleared \\ AvgHeightIncrease \\ NumHoles \\ ColumnTransition \\ AbsoluteDiff \\ RowTransition \\ WellSum \end{pmatrix} \Longrightarrow \begin{pmatrix} -2.9210777318332948 \\ -1.3634453151532058 \\ -6.350233933389025 \\ -3.015076993292611 \\ -7.867411559467035 \\ -1.0057536655180186 \\ -1.6874510272386336 \\ -1.851236997094957 \end{pmatrix}$$

It is interesting to see that the weights obtained for the *RowsCleared* feature is negative, meaning the linear function F minimizes the number of rows cleared instead of maximizing it as one would expect. This may be because it is riskier to allow a greater height without clearing any rows and is a byproduct of height minimization, as there are no incentives for the learning algorithm to allow for such risks (such as a score multiplier when multiple rows are cleared).

7 Results

The results of 100 runs using the weights obtained in Section 6 can be seen in Fig. **??**, and the statistical findings in Table 7.1.

Mean	xxx,xxx
Median	xxx,xxx
Std. Dev.	x.xx
Min	xxx,xxx
Max	x,xxx,xxx

Table 7.1: Statistics of 100 runs

Some commentary on the results here

8 Conclusion

References

[1] F Van Den Bergh and A P Engelbrecht. “A study of particle swarm optimization particle trajectories”. In: *Information Sciences* 176 (2006), pp. 937–971. DOI: 10.1016/j.ins.2005.02.003. URL: https://pdfs.semanticscholar.org/270d/24da13010c775f3b09b9fc53b2e612fc97aa.pdf.