Computational Cognitive Science Project

Task 1

You are required to write and train a genetic algorithm to learn to play the game Tetris. This algorithm can use any means of calculations that do not involve machine learning related techniques such as SVM or any neural network variants. **Although applying genetic algorithms techniques on machine learning models is a feasible and reasonable idea (may even be better… <u>may</u>), it is not allowed for this assignment and will be penalized**.

To focus on the development of the algorithm and not the gameplay, you will be given a class that you can use to evaluate your algorithm. The class handles playing the game mechanics and provides a simulation environment, and the game is a bit simplified in this assignment. Most importantly, the class will give you what you need and requires one thing from you, a function to evaluate the moves it can make. This will be through your algorithm. And you require one thing from this class and that is how well your algorithm performed and it gives you that with a performance score, along with a debug of the states it went through and ratings it got using your function.

Your algorithm will need to include elements of a genetic algorithm. Meaning you must have values (that you will calculate based on your algorithm) that affect the rating of a current move. How much these calculated values should contribute to the rating usually isn't clear (should it positive or negative, should it be more or less) and so you have to multiply them by contribution factor to control their contribution. These contribution factor will be your chromosomes and you have to generate, test and evolve them until you reach to reach a better setting (possibly optimal) if such setting exists (maybe your algorithm needs more calculated values and the ones you extracted are not enough). Note the factors can be negative, positive or even zero (zero meaning your idea was useless). These factors can be in decimal or integer values (does not matter), but they are **<u>NOT</u> limited** to -1.0 to 1.0 (so you can have bigger numbers, just be aware that it can override the effects of a score)

Required Settings

The minimum number of extracted/calculated values need to be at least 5 (figure something out, there are examples given below). The minimum number of contribution factors is 4.

The number of chromosomes should not be less than 12, apply at least 10 evolutions. Make

a log file of the values. Save the optimal values (values that got you the highest score).

Your algorithm needs to win, this is a part of the task and failing to do so will make you not get the grades on this part. Winning here is a relative term, it means to not lose until the end of the run and to try to score the most you can (trying to improve your score). These two things are kind of correlated. You can choose whatever seed you wish, just keep it in the code and **write it in the report**.

Run the game for 300-500 iterations (pieces/plays) for training and for the final test run 600 iterations using the optimal contribution factors found. Write them down in a separate code file and run and **save the score for this run**. Note that the game can end early if you get an early game over.

The game itself

The simplest way to know which game is being referred to is through searching it online, you can even see tournaments each year for it.

The game is simply dropping blocks of different shapes into a board from top to bottom. Each shape is made of a group of squares that are arranged in a certain way, these shapes can be rotated and moved. The simplest space component is a square, it defines the minimum movement and how everything is calculated and made. You earn points by filling a whole line of squares with no gaps. The more lines you clear at once the higher the points you get than clearing them individually or in more steps.

There is a mini easter egg hidden in the assignment.

The given gameplay

You will be given pieces to play and are asked to rate them if they were to fall in each column. According to the first best rating and rotation associated with it, the piece will be rotated to match that rotation and fall down in that given place. The left most part of the piece in whatever rotation is always going to be at the given index being tested (this is guaranteed).

Pieces are generated according to a fixed random seed, so each run will regenerate the same sequence (unless the seed is changed). So only change the seed at the beginning of the running the whole code/script, not between each run to achieve a fair and consistent result.

Rating function

Your rating function is required to do a relatively simple, but a bit complex task. You will be given the current state of the board, the current piece to be played and the next piece, and what column I am thinking of playing it on. You will be asked to return what is the best rotation and its score. You can score each rotation and check who has highest and return it and its associated score. You can also try playing the next anywhere in any rotation (the next piece is not constrained by the column, only the current one).

The reason you are not evaluating all the columns at once for the current piece is that it can eventually be the same. And this will require less code on your part.

Example of Values

The max height in a given state. The difference between max height in this state and the next. The minimum height. The number of gaps (depending on each type of gap). Score gained (if any) for this play. Height of the deepest valley (this is not exactly min height). All of these can be for current state/board and after playing the current piece, and you can even evaluate playing the next piece (remember the next piece can be played anywhere).

You can add whatever calculations comes to mind (Just remember, no machine learning, it needs to be in steps through an algorithm), the above are counting variables or variables that rate something, you should at the end return single rating value based on these variables. Remember that the contribution factors are what control them, and you evolve the factors not the calculated values.

Code

You will be given a code and a **video will be posted explaining how to use the code on google classroom** (along with the code of course). You are required to submit the code file you've used and a final code for the optimal run. Separate the optimal run in another code file. Put in it the same code, without the evolution part and with the optimal found values of the factors and run it for the requested number of iterations (in the final test run mentioned above).

Report

Explain your algorithm in detail and how you think it gets you a better gameplay (this can help if you could not achieve a winning condition). Show the progress of the best two chromosomes in all states (show their score in a graph).

Write the seed you chose, the two best chromosomes/factors and their corresponding score at the end of the run. Also write the score of the optimal factor after doing the final test run (the one that has more iterations, done in a separate file).

You can include any additional findings you wish to share.

Deliverables

- Code
- Report
- Presentation