# Tetris

Jeremy Ku-Benjet

Tetris is a popular puzzle game by Alexey Pajitnov in which the player is given various shaped blocks to place on a rectangular board. The shapes are commonly called $I, T, S, Z, L, J$, and $O$, each corresponding with the similar looking shape.

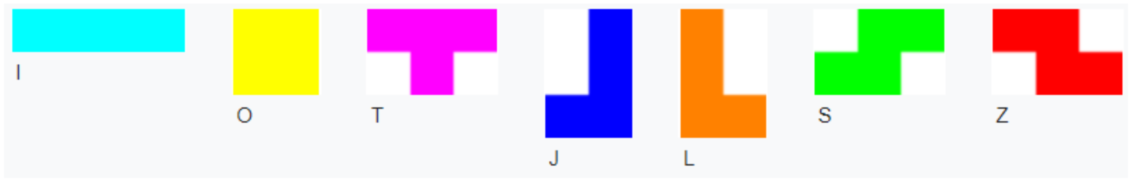

Figure 1: image taken from wikipedia

The player can rotate the blocks or move them horizontally before placing them. When a unbroken line of blocks fills a row, the row is cleared.

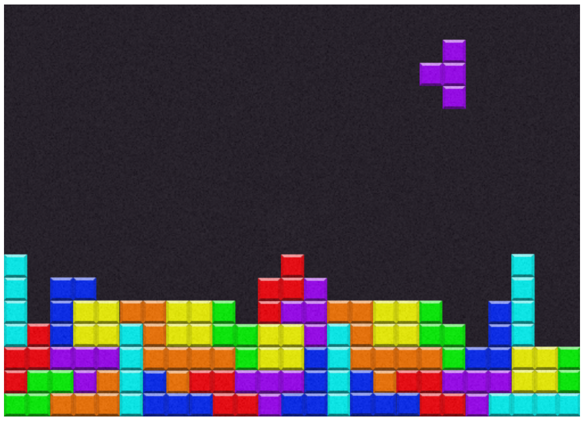

Image by Tiffany Leong

Along with manually playing the game, there has been study into optimal strategies for the game along with its algorithmic complexity.

The first question is what we want an algorithm to do. What is its input, and what is its output? The most natural way to think about Tetris is putting the computer in the same situation we, as the player, are put in. That is the algorithm knows the current state of the board and the next piece to place. This is the "on-line" Tetris puzzle, meaning it processes information as it becomes available. As the algorithm doesn't have all the information at once, finding an optimal solution to a given piece sequence is not always possible. Therefore, the number of cleared lines is often used as a metric for success. [1]

To maximize the cleared lines, a heuristic, called the evaluation function (one of multiple possible names), is used to assign a board a numerical value; a decision can be made based on these values. When playing a game, the board after each possible piece placement can be evaluated and the best move can be determined.

The evaluation function is based on a variety of features of the board, like the current board height or how many covered unfilled squares there are. Each of these features than weighted. This can be done manually or algorithmically. Genetic algorithms and reinforcement learning have both been used to great effect. The weighted features are then added together to assign the board its numerical value. Often, the evaluation function assigns higher values to worse boards. For example, one observed feature may the amount of covered, unfilled squares; the more of which is usually worse. Therefore, the model would place the tile to minimize the value.

Traversing all nodes in the search tree is done with the common graph traversal methods such as breadth first search or depth first search.

[1] *On the evolution of artificial Tetris players*, Amine Boumaza, retreived from https://hal.archives-ouvertes.fr/hal-00397045v1/document

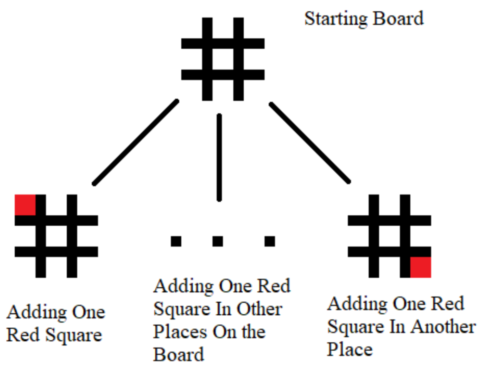
Starting Board
Adding One Red Square
Adding One Red Square In Other Places On the Board
Adding One Red Square In Another Place

Figure 2: Game tree for tick tac toe, red goes first. Image by Me

Evaluating for every possible placement of the known piece gets many possible boards which can be represented in a game tree. A final board is a leaf, or node at the bottom of the tree. Creating a game tree and evaluating each node is standard for many games. If more computational power is available, sometimes the game tree can be expanded to look more than one move in the future, or certain moves can be picked to be look into. However, after the first move the piece to be placed is not known. Testing every possible placement for every possible piece means testing upward of 162 placements. As this needs to be done for every piece placement, the tree grows exponentially and quickly becomes too large for any computer to search.

But while the online Tetris puzzle is most similar to how a human would play Tetris, an entirely optimal solution is impossible as the computer can't foresee its next piece and plan accordingly. To let an optimal solution exist, an "offline" Tetris problem is considered. The puzzle is offline in all of the information, meaning the size of the board and the order of the tetrominos is known at the start of the program.

For example, you might be given a $10 \times 16$ board and the pieces, in order, $I, T, S, Z, O, L, J, O, L, I$ and be asked to determine the maximum number of lines which can be cleared. [2]

An idea to find the optimal amount of lines which can be cleared is to build a search tree out of all possible piece placements. As the entire sequence of pieces is known, this is in theory possible. However, in practice, the tree grows exponentially and a computer which individually follows every path from the top, or root, to the the leaf nodes will never finish. Generally, if an algorithm runs in exponential time it is considered inefficient. Ideally, to make an algorithm efficient, it should run in polynomial time, meaning the number of operations, like adding 1, it preforms is bounded by a polynomial. [3] Problems which can be solved with a polynomial time algorithm are considered to be in P, the set of problems with that property. In fact, many modern day security technologies are built the concept breaking them would require solving a problem not in P.

Back to Tetris, a new question arises: is the offline Tetris problem in P, or as you will more often see: is the offline Tetris problem NP-complete? NP represents a problem which is solvable in non-deterministic polynomial time. Complete is simply saying that Tetris is at least as hard but not harder than all other problems in NP. This is at its core saying that where a standard computer can only try one choice of piece placement for example, this non-deterministic computer can try all of them at the same time. The special non-deterministic computer can run in polynomial time. Another way of thinking about it is the computer is "lucky": every decision it makes is the optimal decision. In the above algorithm this lucky computer would

[2]you can try this puzzle yourself, setting it up with on the human vs. human mode of this website https://ondras.github.io/custom-tetris/ but manually it isn't a super fun puzzle

[3]More detail on what an operation is and more description of time complexity and models of computation is out of the scope of this article. What the complexity of an algorithm is is dependant on the model of computation used.

only have to go from the root to one leaf as it always chooses the optimal piece placement. When it reaches the leaf it knows that the path it took will maximize the lines cleared so the algorithm terminates after only considering each piece in the sequence of given pieces once. As on such a lucky computer, the program would run in polynomial (linear) time, this algorithm is runs in non-deterministic polynomial time.

It turns out that there is no polynomial time algorithm to solve the general Tetris problem! This was proven in 2002 by Demaine, Hohenberger and Liben-Nowell in their paper "Tetris is Hard, Even to Approximate"[4] and again, with a simpler argument, by Breukelaar, Hoogeboom, and Kosters in their paper "Tetris is hard, Made Easy"[5]. Both of these arguments center around reducing Tetris to the 3-Partition problem, another NP-complete problem. [6]

The 3-partition problem gives a sequence of $3s$ positive integers and a positive integer $T$. The sum of the sequence of integers is $sT$ and each individual integer in the sequence $a_i$ fulfils $T/4 < a_i < T/2$. The question asks if all the sequence of integers can be divided into $s$ subsets such that the sum of each subset is $T$. Note, as $T/4 < a_i < T/2$, each subset has exactly three elements.

To reduce Tetris to 3-Parition, an algorithm which outputs information about a game of Tetris must be able to tell us an answer to a 3-Parition problem. When analyzing if the board can be cleared, if the board can be cleared, that would correspond to a "yes" to the 3-Parition problem and if the board cannot be cleared, it would correspond to a "no". To do this, the 3-Parition question is constructed using the tools of Tetris.

Tetris however does not have numbers to divide into groups, only falling pieces, so the 3-Parition problem must be reinterpreted. First, consider the all the $a_i$ in the input in unary (base 1). Each number is now a bunch of tick marks, meaning adding two numbers is just combining tick marks. For example 10 is 1111111111 and 7 is 1111111 so $10 + 7 = 17$ is $1111111111 + 1111111 = 11111111111111111$. As all the numbers are in unary, each set with sum $T$ can be thought of as a "bucket" which holds $T$ tick marks. To add a number to a set, add all of the tick marks to the set, to add multiple numbers, do that multiple times until the bucket is full. Representing numbers in unary is often a useful tool for reductions with the 3-Parition problem.

Thinking of each number as tick marks and each of the $s$ sets as buckets, building the 3-Parition problem in Tetris becomes feasible. Each tick mark can be represented by a sequence of pieces in the game: first a piece to denote the beginning of a number, a number of pieces, each group denoting one digit of the unary number, and finally a group of pieces to represent the end of the number. Each bucket can be represented by formations of squares in the starting board made only to accept entire numbers. The image above shows a construction using and $L$ piece to start to start the number and repeated $O, J, O$ pieces are used for each tick mark. To end a number an $O$ piece followed by an $I$ piece is used. If the unfilled volume of the buckets is the same as the volume of the pieces denoting numbers then for the entire board to be cleared, each bucket must be completely filled.

If there is an algorithm which could tell if a board could be cleared or not given a starting board then it can tell if all of the buckets can be filled given a sequence of numbers and solve an equivalent 3-Parititition problem. As the 3-Parition problem is NP-Complete and an algorithm which solves this special case of Tetris will give an answer to the 3-Parition problem, that algorithm would also have to be NP-Complete. As determining if a board can be cleared is a special case of determining the most possible lines which can be cleared, finding the most lines which can be cleared must also be NP-Complete.

Unless P is the same as NP, this means that an algorithm which efficiently solves the general Tetris problem proposed is impossible. However narrowing the scope of the algorithm we want, limiting the amount of rows or columns, makes constructing an NP-complete problem with the starting board impossible. What seems like such a strong claim about the generalized Tetris game, quickly breaks down as we remove the difficult cases which are used to prove it is Tetris NP-complete. For some limitations on the board, the complexity of the Tetris problem is unknown.

Even if we can't play Tetris optimally in general, a method to efficiently play Tetris on a clear board may exist, but until then, modern artificial Tetris players can clear millions of lines on average. Despite it

[4]retrieved from https://arxiv.org/pdf/cs/0210020.pdf

[5]retreived from https://www.cs.montana.edu/courses/spring2004/513/resources/BH03.pdf

[6]While I try to give the gist of the argument, it is impossible to give the full argument. I leave out an important part of the argument for simplicity as well as don't prove many statements. To really understand this proof should read the papers referenced.

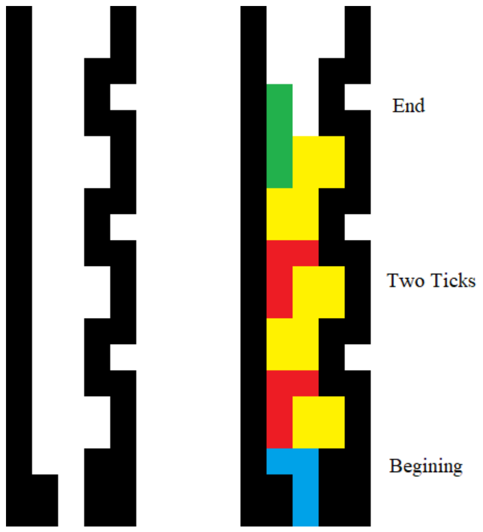

Figure 3: Example bucket, construction from Breukelaar, Hoogeboom, and Kosters, image by me

possibly not being optimal, at least I find that still pretty cool.

## References Further Reading

- Giant website of tetris by Colin Fahey: https://www.colinfahey.com/tetris/
- Not related to Tetris, but a nice article on showing a different game NP-Complete, by Richard Kaye: http://simon.bailey.at/random/kaye.minesweeper.pdf