Most scales are built upon tetra-chords, but what if we used Tetris-chords

Written and programmed by Kevin Lyvers

Introduction

Iannis Xenakis used game theory concepts in three of his compositions and he was one of the first notable composers to formalize these math concepts into music. He argued music could be thought of as a set of choices that a musician or composer can make, influencing the music in a similar way as to how a player would influence a game of chess by moving a piece. The field of game theory is very wide and flexible with the topics being able to be applied to pitches, amplitudes, structures, articulations, or any other dimension of music with a myriad of control mechanisms. Whereas Xenakis used game matrices to provide structure to his instrumentation, I have taken the field of game theory more literally - implementing music to an actual game.

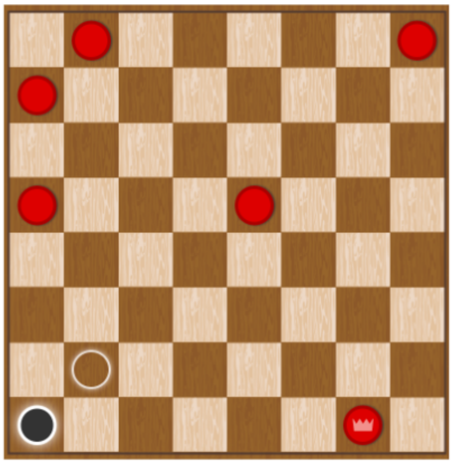
Currently housed at https://kevinlyvers.com/info/[1], I have created a playable version of tetris where the background audio is directly affected by the current game board state. The "worse" a certain current game-state[2] is, the more dissonant and chaotic the audio will become. This process can be thought of as two large components: evaluating a tetris game-state, and producing music with a certain level of dissonance.

Evaluating Tetris

Originally this project was going to focus around a game of checkers. Checkers is nice in the fact that it is a game where all the information is available. A strong enough computer could look at a game-state of checkers and compute all possible outcomes following a particular move, knowing exactly how good any move is and all possible outcomes thereafter. This can allow us to characterize how good any checkers board is for a given player by simply looking ahead. Some game-states are obvious, but others require millions of computations. However, I simply did not care about checkers, and quickly got bored. I then moved to Tetris.

[1] It should be playable there for a long time. If it is not and you would like to play it, reach out to me.
[2] A game-state is a snapshot of a game - a singular board position we can analyze.

A fairly even board-state
for both players


A board state where red
has a strong advantage

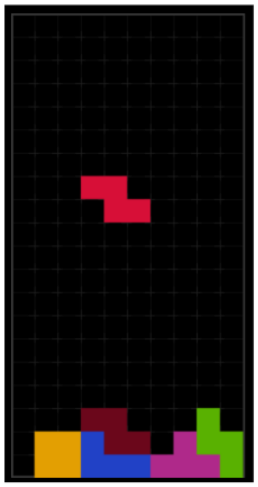
Tetris comes with one major major difference from a game theory standpoint from checkers. Tetris is not an easy game to look ahead. In some Tetris games we can look ahead five pieces, which is more information to use to plan the future, but the best places to place those pieces are not always clear. A certain piece might be okay with the current situation, but in an unpredictable ten pieces it could lead to a game ending situation. As with life, the future in a game of Tetris is unknowable, so we must create our own metrics to define a board state not based on future outcomes.

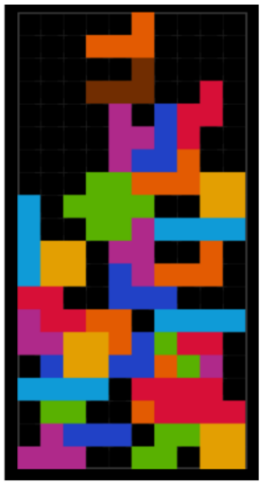
Previously I created a Tetris-bot that was based around a similar method of calculating how good a given game-state is using predefined measures. Obviously, a board that is stacked very high is less favorable than a very low stack, because the higher stacks are closer to a game-over. I plugged in my previous equations, but I was unsatisfied. I have grown in intelligence, and it's fitting my equations should also become smarter. I did the smart thing and consulted other literature.

Researchers at The National University of Singapore tried to get a similar metric in Tetris in order to clear a maximum number of lines in a given game of Tetris[3]. They ran a genetic algorithm over many iterations of a Tetris evaluating weights for:

1. Sum of height differences between adjacent columns.
2. Maximum height of a column.
3. Total number of cleared rows.
4. Whether the current move results in a loss.
5. Total number of holes.
6. Sum of depths of all wells in the game board.
7. Mean of the absolute difference between the height of each column and the mean height.

Metrics 3 and 4 are not very applicable to my evaluation, because I am looking at a given game-state, not a hypothetical placement of a future piece. However the other five metrics are implementable and useful. They are typically characterizing what we would think of as how "wavy", "holey", and "tall" a certain board is. "Wavy" and "holey" boards make a flat placement of pieces more difficult, while a "taller" board makes for an eventual gameover.


A favorable
board position


An unfavorable
board position

[3] https://theultramarine19.github.io/data/Tetris.pdf

Using the researcher's found weights of $\hat{W} = [0.69, −0.60, 0.68, 1.13, 3.63]$, I applied the metrics to a linear equation where the inputs are the five characteristic of the board. This allows us to look at the current game-state and assign it a specific, one value of how favorable it is. A $\hat{W}$ value of 0 is a perfectly clear board, which is as safe and good of a state that there can be. Simply from play testing bad games a $\hat{W}$ value of 30 is an awful state that will surely lead to a gameover. This allows us to gauge a level of disorder in the game. This would allow a normal game-theorist to look at optimal outcomes to minimize $\hat{W}$ to keep a game progressing, however we can use it to control sounds. We can quantify a game-state with one number.[4]

Producing dissonance

With avant-garde and modern musicians and composers existing, no music can really be called ugly or harsh anymore, but for the matters of this project I will be thinking very westernly and diatonically. Chordal tones are favorable. A major triad and such create an easy listening texture compared to harsher intervals.

I downloaded 24 piano audio samples[5] that each were simply a note ranging chromatically from F_3 to E_5 held for about a second. Pursuing pre-recorded audio samples limits me in some aspects like dynamics and articulation, but audio synthesis would have been too costly computationally and timely for a web project. Since the lowest note was an F, I determined the program would center around F major. I then subjectively ranked all the notes based on how "calm" they sound. My subjective ordering was:

[I,V,III,IV,VI,bVI,bVII,II,bV,VII,bII]

ranked most pleasant to least pleasant given an F bass, with similar rankings for the octave above. Once again, I acknowledge it is very subjective, but as the composer/programmer I was allowed to make these subjective decisions.

[4] The corresponding variables and singular output number all appear at the top of the game while playing.

[5] https://archive.org/details/24-piano-keys/key21.mp3

To create less dissonant music we only access the first elements in the ordering, while for more dissonant music we open up to later entries. This creates more dissonant music to have harsher intervals and wider random jumps as we access more elements from the array, while less dissonant music is many chordal tones with minimal large jumps.

Another element of chaos is the timing. Programming precise timing is above my knowledge on web-based programming so there are elements of randomness in the timing. To create less chaotic timing, I limit tones from sounding only very close to every second, while more chaotic timing will let a sound play whenever they execute their commands. By the nature of non-strict random timing, allowing a sample to sound whenever they run will naturally cause chaos. The culmination will be clusterful and unpredictable, while the opposite of closing the time gap of when they can produce creates the illusion of a consistent beat with less randomness.

Combining

We can then combine these two features. We have an equation that gives us a number based on how favorable the game-state of a Tetris board is, and we have an equation to generate a certain level of chaos in music given a threshold. We simply link these two to create a tetris music generator.

The largest thing of interest is the trial and error it took to get the correlation how I wanted. I wanted the music to become exponentially more chaotic, leading me to create several exponential equations until I got the fit I wanted. For a tetris game board factor $\hat{W}$ and and music dissonance $\hat{C}$ the correlation ended at $(\hat{W}/2)^{2.2} = \hat{C}$. This created a combination I liked where the music kept calm at early stages with it dramatically ramping towards higher stacks.

Conclusion

This has now become a functioning music creation tool - admitally, one with very little control of how it sounds. There is a similar, more fleshed out video game of this called *Tetris Effect,* where the music is controlled by what you do in the game. However the music is prerecorded and it is just the speed of placing pieces that affects the tempo and beat of the music.

It allows for an immersive musical Tetris experience, and as a true Tetris lover, I believe it is one of the most amazing things to come out of video games in the last twenty years.

My project is more experimental. Without a high level structure realizable because of the instant changing nature of Tetris, most pieces composed are strange. Most do follow a typical accelerando as your game falls into chaos. I have recorded myself playing a game with composition in mind - timing my pieces and using my knowledge of the program to create a composition. It can be found at https://kevinlyvers.com/info/composition.mp4.

Postface: In depth-programming and Tetris

I think to include the technologies and implementations in the main part of the paper would distract from the conceptualization of my project, so I decided to include them back here.

This was all made in a Javascript framework called P5js. P5js is my go-to for proof-of-concept projects, because of its ease and avid fanbase. There are plenty of tutorials and helpful creators out there giving advice on how to implement games, visualizations, or applications using P5js. The largest downside to the framework is its handling of audio which caused my limited 24 note inventory with no dynamic differences. P5js also entirely runs from a web-editor which typically saves my laptop from catching fire from my IDE, VScode. Javascript comes with its own hassle, but there's nothing Github Co-pilot can't solve - and at least it's not fortran. I knew I wanted to write a web-based program rather than Python or Java, so readers can interact with my conceptualization with minimal effort. They just type in the URL and it will work for them.

Additionally, I worked hard to implement a "true" Tetris[6] game. As previously said I am a truly huge Tetris lover, so I wanted to do it right. I implemented the correct colors for each block, with their correct SRS rotations. I considered adding the now-typically one piece look ahead, but the implementation would have been much too hard. The pieces do spawn in true Tetris 7-bag randomizer methods, and T-spins are possible. I took time to make this a realistic version of Tetris even if almost all wouldn't notice. The largest exception is that you cannot lose in my version. If you fail and "top out," the game will keep going. It was at first a bug that became a feature, because then one would not have to worry about playing well to make music. One could be the worst player and still make a piece as long as they would like.

Funnily, the original Tetris theme song was in the key of F, just like my program.

[6] I thought long and hard whether or not to write Tetris or *Tetris*, but I decided with the unitalicized variant, because most of the time I am talking about the kind of game, not the specific copyrighted distribution of the game.