Universitat
de les Illes Balears

FINAL DEGREE REPORT

TETRIS NEURAL NET PLAYING IN THE NINTENDO SWITCH

Joan Dot Sastre

Degree in informatic engineering

Escola Politècnica Superiorl

Academic Year 2020-21

TETRIS NEURAL NET PLAYING IN THE NINTENDO SWITCH

Joan Dot Sastre

Final Degree Report

Escola Politècnica Superior

Universitat de les Illes Balears

Academic Year 2020-21

Key words: AI, deep learning, deep q learning, tetris

Tutor: José María Buades Rubio & Gabriel Moyá Alcover

Autoritz la Universitat a incloure aquest treball en el repositori institucional per consultar-lo en accés obert i difondre'l en línia, amb finalitats exclusivament acadèmiques i d'investigació	Autor/a Sí	Autor/a No	Tutor/a Sí	Tutor/a No
	☑	☐	☑	☐

Thanks to José María Buades, for suggesting such a nice idea of a final degree project
and helping me throughout all the steps whenever i needed.

CONTENTS

ACRÒNIMS

AI Artificial Inteligence

PC Personal Computer

API Application Programming Interface

SRS Standard Rotation System

NPC Non-Playable Character

PROJECT SUMMARY

The basis of this project consists in achieving an Artificial Inteligence (AI) capable of playing the game “Tetris 99” in the console known as “Switch”, manufactured by the famous game company Nintendo. The task may seem simple at the beginning, but the sole nature of having to intercommunicate two devices (PC and console), with non-existent tools comercially available, already shows us that this will not be a trivial matter. The AI has to be built and trained through a custom PC environment, and then be able to receive and send information to the console reliably.

CHAPTER 1

INTRODUCTION

1.1 Artificial intelligence in video games

AI has been present in video games since the very beginning. Its purpose has always been to improve the players experience and the methods that have been used to implement such behaviours are vast, ranging from finite state machines and increasingly more complex enemy movement patterns tied to the game difficulty/level, to combining different advanced methods like pathfinding and decision trees. Other techniques related to machine learning such as reinforcement learning can currently also be found in some games. All these methods are mostly used for Non-Playable Character (NPC)s and the information they perceive from the environment can be given in two different ways, via sensors, which provide a limited vision of the game world, or via the game's own stored information e.g., the player's exact location.

Due to an increasing interest in artificial intelligence in recent years, people have started to try and beat their favourite games with it. When taking this approach, we must first consider how the agent is going to perceive the game, having the same two options we talked about before. This time we usually encounter a major inconvenience, we do not have direct access to the game information due to us not being the game developers, although thanks to some Application Programming Interface (API) (such as OpenAI Gym) we can access the game and thus base our agent's information on it. Unfortunately, those APIs mostly feature older games, which limits us to the ones provided by it. Hence comes the need for image processing tools to extract data, though this may not necessarily be done by us, as will be shown later.

Due to the increasingly more difficult games being beaten, has also come a need for more intricate agents, leading to the drop of simpler techniques in favour of reinforcement learning (many times paired with those old techniques in order to provide the agent with basic behavioural guidance). This has ended up providing much better results than previously achieved in highly complex environments, and also helped discover new strategies in the own game. Even exploits in the system have sometimes been found, like in the case of an OpenAi project, where in a hide and seek game, the

agents managed to abuse the physics engine in various ways.

1.2 Objectives and setbacks

The overall goal of the project has already been discussed, but what will be called a success has not yet been defined. Building an AI capable of playing Tetris has already been done many times before with great success, although the challenge trying to be taken on has a few more major and minor hindrances.

First of all, as a minor inconvenience, the Tetris version we are building our AI on features the Standard Rotation System (SRS), which is a modern rotation system with some unconventional situational rotations. No implementation that can be used has been found, neither as an OpenAI gym nor as simple game. Because of this, an entire game replicating Tetris 99 must be built from scratch to train our model.

Secondly, there is not a standardized way to access the console's controller port from a PC, so a reliable workaround must be built and adapted. This is probably the biggest setback.

Lastly, and as a result of having to intercommunicate both devices, some extra delay, that we hope will not heavily interfere, will occur when bringing everything together.

Having mentioned the setbacks we first encounter, we expect to build an agent that plays tetris to near perfection, never loosing a game and trying to make as many points as possible in the least amount of time. If anything, we expect only the conditions outside the agents power to make it perform badly, namely bad screen detections or missed inputs. This means that once a good enough agent has been built, our main focus will shift onto making it perform as closely as possible to its intended actions on the console.

1.3 Task division

In order to reach our goal, we have to tackle the problems one by one. Thus, the means by which the results in the project have been obtained consist of dividing it into four different modules:

- Switch-PC interface: The way in which the pc is able to communicate with the console.
- Information capture: How the console's information is sent to the pc and then processed for use by the neural net.
- Machine learning: How the AI is able to learn. Includes the training environment explanation, the heuristic used and how it was chosen.
- Decision making: Defines how the information extracted by the information capture module is treated right before it is finally ready to be sent to the net. It also explains how the output is adapted and transferred to the console correctly.

The aforementioned modules will be further explained later, in their corresponding section in the document but, it needs to be mentiones that the whole project has been made in python given its many machine learning library options.

CHAPTER 2

TETRIS 99 AND SYSTEM BUILT

2.1 Tetris version

Tetris is a long running game series that has been ongoing since 1984, when Alexey Pajitnov invented it. Ever since it was created, many iterations of the game have been made, with each one of those somewhat altering the rules or adding new mechanics to spice things up. As previously mentioned, the project is being made under the Tetris 99 version, which implements the SRS. This version has been chosen due to it being the most modern tetris up to date and because of the challenge of having our AI work on another platform.

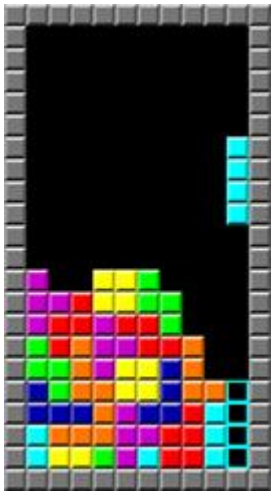


Figure 2.1: Tetris grid example

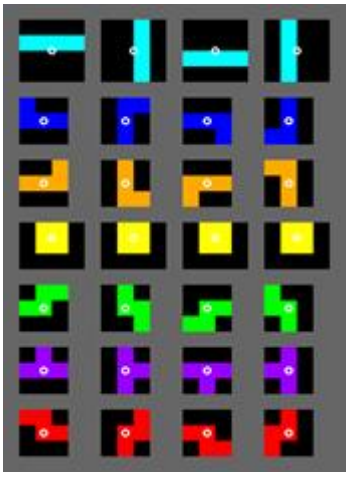


Figure 2.2: Tetris pieces and their rotations

Single	100 × level difficulty
Double	300 × level difficulty
Triple	500 × level difficulty
Tetris (Quadruple)	800 × level difficulty

Table 2.1: Tetris score by lines cleared

Combo	50 × combo count × level
Soft drop	1 per cell
Hard drop	2 per cell

Table 2.2: Tetris score by movement and combos

2.2 Game basics

As many people already know, Tetris is a puzzle game consisting in trying to stack pieces up pieces and clear lines on a 10x20 grid. Whenever a line is filled to its maximum capacity it gets cleared and the blocks above it drop as many lines as were cleared. Whenever a piece is locked in place in an altitude higher than the game grid plus one you lose. A grid could look like figure 2.1.

There is a total of 7 different pieces, each one of those having an associated colour that is usually maintained through all Tetris versions. Their names are I, J, L O, S, T and Z as seen in figure 2.2.

As we can see in the image just referenced, each piece has four different orientations which can be accessed sequentially back and forth in the order shown, the small circle indicating the axis the piece rotates in.

The I and O pieces are a special case given that they do not use an actual block as their anchor point to rotate, making the first one shift one block up or down depending on the current position and the second one not rotate at all.

Now that we know their shapes, we see that the maximum number of lines that can be cleared at once is four. This is crucial because the score we obtain does not increase linearly, netting us higher scores the more lines we clear in one go.

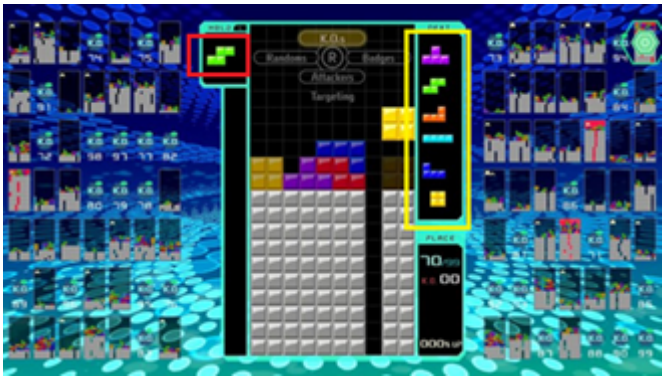


Figure 2.3: Actual Tetris 99 game

The actual formula which dictates how many points we get can be seen in 2.1

More ways of obtaining points are soft drops (moving the piece down one cell), hard drops (letting the piece fall to the bottom) and combos (chaining line clears with different pieces), which go as 2.2

Finally, there is T-spins, which is a mechanic that will be spoken about at the end of this block, once all the information surrounding the SRS has been laid out. The scoring system just described works on single player game modes, but in Tetris 99, the main game mode actually involves 99 players concurrently battling against each other. Here line clears serve another purpose, sending "garbage lines" to the opponents. More on this system in the following section.

2.3 UI and specifics

An actual game of Tetris 99 will look like fig2.3. There are many things that need to be analyzed to fully understand all the game aspects of this game version.

The first thing that should be mentioned is that we can see the upcoming 6 pieces that will have to be placed on the board (highlighted by the yellow square). Those are chosen from a bag containing all seven pieces, therefore making the game much more predictable as luck will not be impacting the game as much as with a fully random selector.

Then, there is the piece storage block (encapsulated by the red square), which allows us to save the current piece and draw the next one or, if there already is a stored one, to swap it out.

Last and more importantly, the background shows many more smaller Tetris boards, which belong to other players who are competing against you. In this mode you cannot see your score, and your performance is based on surviving the longest. When clearing lines, you now send garbage lines (grey blocks) to whoever of those players you are targeting:

- Clear two lines: Send one line of garbage
- Clear three lines: Send two lines of garbage
- Clear four lines: Send four lines of garbage

- Clear the full board: +4 lines of garbage

Whenever you kill a player, a part of a badge is awarded to you. Each badge is increasingly more difficult to get, and you can only get up to four in total:

- Two knockouts: 25% garbage bonus
- Six knockouts: 50% garbage bonus
- 14 knockouts: 75% garbage bonus
- 30 knockouts: 100% garbage bonus

It may seem quite difficult to complete all badges, but the method is eased by being able to steal the badges from a player you have defeated. You can choose between five attacking modes to target different opponents:

- K.O.s: targets whoever is closer to losing the game.
- Randoms.
- Badges: targets whoever has more badges.
- Attackers: targets whoever is attacking you.
- Choice: manually select a specific player.

If you are targeted by multiple opponents, a boost to attack power is received:

- 2 Opponents: +1 Bonus lines sent
- 3 Opponents: +3 Bonus lines sent
- 4 Opponents: +5 Bonus lines sent
- 5 Opponents: +7 Bonus lines sent
- 6+ Opponents: +9 Bonus lines sent

This boost is applied before the badge attack boost.

It should also be noted that when receiving garbage lines, those will first be shown in the column right under your piece storage, and only be added to your board after some time. The time is indicated by 3 colour stages, being grey, yellow, and red, from best to worst. Garbage lines can also be cleared before they are added to your board by simply clearing lines.

Figure 2.4: No kick phase

Figure 2.5: Right kick phase

Figure 2.6: Up right kick phase

2.4 SRS (Standard rotation system)

Now we can focus on the most intricate part of the game, the rotation system. The basics of this system have already been mentioned, however there is a much deeper pattern to it, which allows us to rotate pieces into places we would not normally be able to. These situations occur when a rotation that is not possible because a collision is detected, and the system tries to move the piece into four different offsets sequentially, sticking to whichever one works first. There are mainly two kinds of offsets, the ones that straight up ignore some collisions and allow you to rotate passing through blocks, and the ones that move you to another location. When an offset displaces you, it is known in game terms as a "kick", and it should be noted that kicks can be performed against walls and pieces equally, propelling you in on or even two directions at the same time, even upwards. Because of the existence of upward kicks, a system limiting the number that can be performed had to be implemented to avoid infinite stalling.

As there is a very large variety of rotations and kicks that can be performed, only a few examples that represent most cases can be seen in figures 2.4, 2.5, 2.6, 2.7.

Now that we have showed some examples, we can talk about T-spins. As its own name implies T-spins are performed using the T piece, and they happen whenever we manage to offset the piece into clearing 1, 2 or three lines, giving us 2, 4 and 6 garbage lines/800 × level, 1200 × level, 1600 × level respectively.

Figure 2.7: Down kick phase

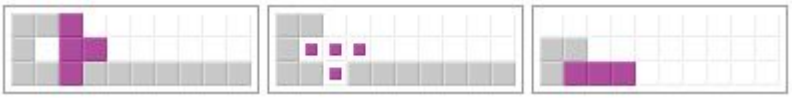


Figure 2.8: T-spin single

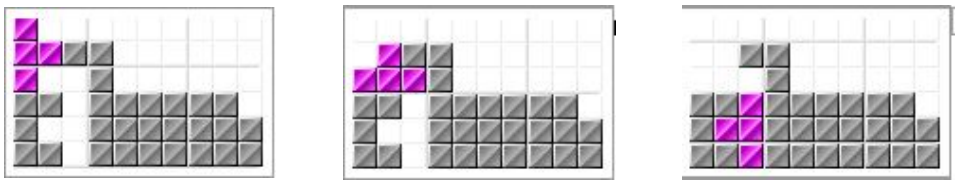


Figure 2.9: T-spin double

2.5 Our system

Having defined all important aspects of tetris 99, an environment similar enough had to be built in order for the AI not to find itself in a foreign situation when being tested in the Nintendo Switch. Thus a search for similar looking tetris implementations had to be done. Unfortunately, as mentioned when talking about the project setbacks, no complete candidates where found.

Thanks to the information found regarding how the SRS works and a project giving an example implementation in C#, we built from scratch a similar looking system that fulfilled our requirements. The system is constituted by three classes, board, piece and tile, each one being controlled and used by the one mentioned before it. Finally, the the script called "tetris.py" unites all of them together and creates a visual interface to play on and show the net's output whenever requested.

To explain the following data structure a bottom up approach will be used in order to follow the thought process behind its implementation.

2.5.1 Tile

Tile is a class very short class containing a position in two dimensions (x,y) and a colour in rgb format. It also includes a method that given a center point and a direction, rotates its own coordinates once.

2.5.2 Piece

A piece object is consituted by "n" number of tiles depending on the piece type. There is a piece child class for each possible piece type and every one of them contains a list of

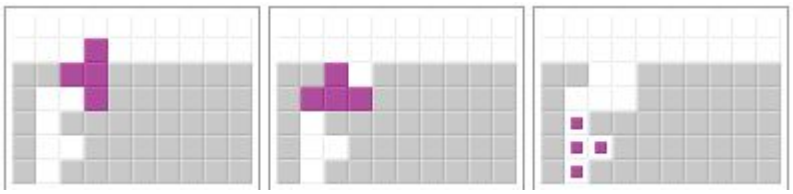


Figure 2.10: T-spin triple

each of the displacements or kicks that can be performed given a rotation. The method rotate piece loops through each tile calling their own rotate method using the center tile as the anchor.

2.5.3 Board

Board is the biggest class of the three consituting the data structure. It is in charge of implementing all the tetris 99 rules there are. Here there is also a piece rotating method, which is in charge of checking wether a kick can be performed or not, given it has the position of each of the blocks and the walls. Despite having implemented the t-spin moves a system that puntuates them has not been built, as it will not be used to reward the neural net (more information on that in the neural net section).

2.5.4 Tetris game

Finally, we have the tetris script, coded using pygame. There is two different implementations within it which will be used to our convenience. The first one calls the main game loop and allows us to play the game normally, reading our keyboard input and calling the appropiate board methods to perfrom each one of them:

- ↓: "Soft drop", drops the piece down by one line.
- ↑: "Hard drop", drop the piece as far down as it can.
- ←: Moves left by one column if possible.
- →: Moves right by one column if possible.
- space bar: Stores the current piece if a piece was not just stored.
- "a": Rotates left.
- "d": Rotates right.

The second one is used by the neural net to show each of its moves. This means that the board game object is manipulated solely by the AI and the scrpit is only used to render the visual information of it by calling the applicable methods.

Either way, both can be stopped by exiting the window clicking the top right "X".

CHAPTER 3

SWITCH-PC INTERFACE

A key aspect to the project consist in how we connect the Personal Computer (PC) and the console. The first part of it, getting the images from the game, has already been dealt with in the previous chapter, but now comes the challenging part. We need to somehow make the neural net's output get to the console.

3.1 Approaches and final solution

As mentioned in the introduction, there is no way to control the Nintendo switch besides using its own controller. The first thing that came to mind was building a robot capable of pressing the buttons itself whenever we told it to. A rough way of implementing it was thought about and some actual piece candidates where found, but nothing came of it, as the difficulty of building such device could be a final project of its own.

Parallely, a way of faking the pc as a controller was investigated. A project that could record input onto an arduino and then send it to the switch by connecting it directly into the console's controller port, was found, which told us that accessing the console was possible. This works by using the same checking protocol sequence as a Nintendo controller when trying to connect the device.

By modifying the project, we allowed for it to instead of record some input sequence, receive it from the PC. As the project comes with a python script that contains the whole controller scheme and methods to test and ensure a good connection, we can plug the script directly in our project for its future use.

3.2 Arduino tools

The original project only used the arduino seen in image 3.1, which already had the commands imprinted in a loop. This time around, as it is connected to our PC using

a USB-TTL converter (.4), seen in 3.2, the loop waits for the input and sends it to the console.

The USB-TTL converter must be connected according to the information given by the own device, which means that contrary to what is most common, RX to TX and TX to RX, it is done backwards. This is only because the converter tells us where the connection must go and not what it actually is. As for the ground and 5V cables there is no room for doubt.

Figure 3.1: Arduino ELEGOO UNO R3

Figure 3.2: USB to TTL CP2102

CHAPTER 4

INFORMATION CAPTURE

If we want to play on the Switch, a system that allows us to decrypt what is on screen, to feed it to the neural net is needed. Our first approach was to use some sort of video device, like a webcam, to record what was on screen while simultaneously sending that information to the PC. That idea was quickly scrapped, as using a normal capture card was the obvious easiest go to. Thanks to that, we can use Opencv (see more in .1) to get the images directly in real time with minimal delay in order to perform whichever operations we need to. OpenCV was chosen as our image processing tool mainly because of its vast number of examples and information regarding it.

4.1 Detection

As previously mentioned, a game of Tetris 99 looks like the image in 4.1:

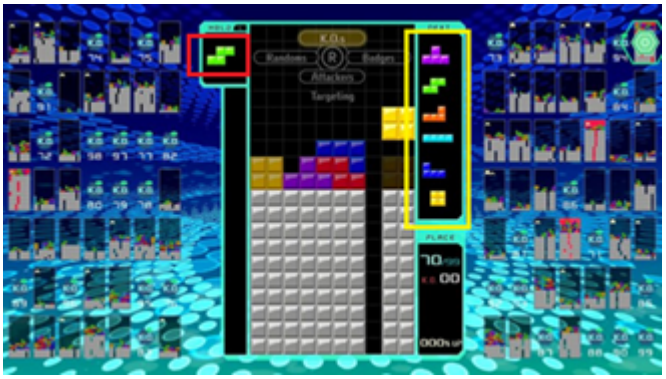


Figure 4.1: Highlighted detection areas

Given that the most important information we need is what are the pieces currently in play, a method to detect whether there is one and which type it is had to be devised. At first it was thought this could be guessed by the piece's own shape but, given rotations

and having access to the colour channel, a detection by the later method was much faster and easier to do. Thus, came the idea of calculating the mean colour of each shape in a 5×5 matrix from its center point, including empty blocks and grey pieces. Still, as seen in the previous image, when a piece is placed, it turns a darker tone of its former colour, making us have to factor those in. Thanks to this minor inconvenience, we will later be able to distinguish the main piece from an already placed one if we need to do so. Finally, a small leeway to the mean colour of each piece had to be taken into account when checking for matches due to other elements in the game board influencing the colour of the own piece with shines or shades.

The information we get is also presented back to the user in real time by drawing the conclusions on top of the processed frame. This helps us understand what is being detected and therefore being fed to the neural net. Depending on the detection a string matching the detection will be shown:

- "e": Means empty block.
- "S", "Z", "I", "T", "J", "L", "O": Refers to each of the possible pieces found in a tetris game.
- "gr": Means grey block.

There is one more element that will not be shown as a letter, "No match", which will be displayed using the last letter found in the block in white, otherwise their respective piece colour or black for the empty will be shown.

4.1.1 Game grid

The first element we try to detect is the game grid. It can be done thanks to having manually found where the cells center pixel is, and how wide and tall each cell is. By applying a for loop, we can then iterate through each cell and store the information in two arrays with the size of the game grid (10×20). The first array contains 0's for empty cells, 1's for blocks and 2's for the main piece (game_matrix), while the second one contains information related to the colour, including a "no match" variable (info_matrix). On each iteration, game_matrix block will be updated only if a match was found, else it will assume the board's state has not changed.

4.1.2 Out of the grid

The next element we detect is a line upwards out of the main game matrix (row 21). This must be done due to the main piece spawn position going up when it cannot be placed at a certain height, the maximum being 21. This was done separately due to the background colour not being black and because only the main piece is displayed at that height, with placed blocks being hidden by the borders 4.2, 4.3.

4.1.3 Check stored piece

To detect which piece (or if none) is in storage, a system that casts two rows of three detection points was built. Each one of those points corresponds to a possible place of

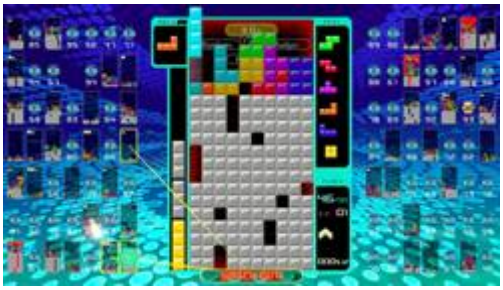


Figure 4.2: Main piece out of grid

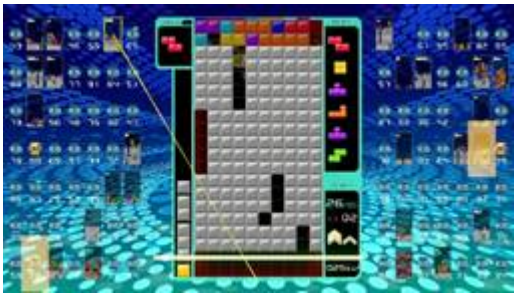


Figure 4.3: Blocks out of grid

a block and, in case it is filled, it updates two matrices with the same system that was built for the game grid detection.

4.1.4 Check upcoming pieces

In order to know what is coming up next, we reused the system to detect stored pieces, this time iterating through a for loop once for each of the upcoming six pieces. Each of the matrices detected is then stored in an array in the same order they are detected (top to bottom).

4.2 How noise affects detection

As mentioned before, we cannot differentiate piece colours without adding a small margin of error for the detection to be consistent in the majority of cases. Unfortunately not only are the colours influenced by other elements, but also visual effects spawn all across the game board depending on the actions done.

To begin with, there is an effect that tells us where our piece is going to be placed (see figure 4.4). Luckily this feature was found to be able to be turned off in the game options, although it is the only one that can be filtered out this easily.

As for the other effects, things like red screen borders, arrows pointing other players and glitter when dropping a piece or sending grey blocks to opponents can also be found interfering with detections (some also seen in 4.4). Many of those directly block what is behind them, so no image processing or margin can be set to minimize or eliminate the obstruction. The best solution we came across is not updating the board information whenever a foreign object is detected, which ends up working pretty well as many of the effects disappear pretty quickly.

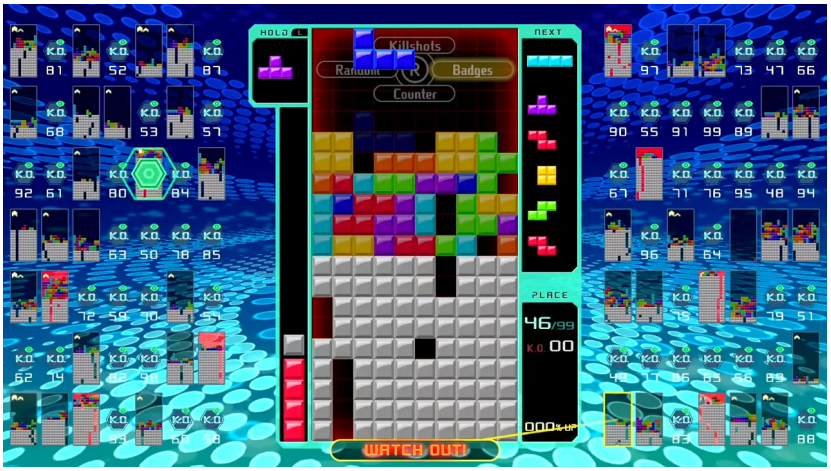


Figure 4.4: Game effects

4.3 Information adaptation for the neural net

Once all the information needed has been collected, a way of adapting it so that the neural net understands it had to be made. To do so, we had to convert the data into a Board object of our own Tetris game, with the same configuration detected.

As our neural net works by finding the best combination of consecutive rotations and horizontal movement, and then dropping the piece, we only need to ask for the neural net's response once (when the piece just spawned). This way we only try to detect new information when the last sequence of movements is completed and when a new piece is detected at the top five rows of the grid, which are the only rows a piece can spawn in plus two more, in case there is a late detection.

When a new main piece is detected, which as mentioned before can be done thanks to it being able to be told apart from the rest because of being slightly brighter, we construct a Piece object with its type and position. The type can easily be guessed by just checking the first blocks's colour but the position is trickier. As all pieces always spawn in the same exact rotation and x position, we can always assign it to be the same, "4", as for the height, we it will be the one of the first block found -1, given that each piece's center block is always one row down except for the "I" piece.

We can now proceed with the creation of the game grid that will be added to the Board object. It is quite simple to do so given that we have a 10×20 array stored with information regarding each cell. We now only have to filter out the main piece, add four empty rows at the top and reverse it to match the object's model. The actual colour of the placed blocks does not matter, as it is merely an aesthetic element.

Finally, we check if there is a stored piece. If positive, we must then check its colour to know if it is an option for it to be placed or not. That information is then passed on to the Board object.

CHAPTER 5

DEEP LEARNING MODULE

CHAPTER 6

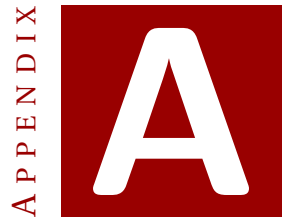
DECISION MAKING

CHAPTER 7

RESULTS

CHAPTER 8

CONCLUSION



APENDIX

.1 Opencv

Opencv is...

.2 Tensorflow

.3 Arduino

.4 USB to TTl

A USB-TTL converter mediates between the rather complex signalling that USB uses, and a simple serial bit stream, using TTL voltage levels and asynchronous communication (ASCII). With the proper drivers and a USB-TTL converter installed, a computer can use any sort of “tty” comm protocol or app to talk to an arbitrary serial device (e,g, an Arduino or a GPS unit).