**BACHELOR’S THESIS**

**TETRIS NEURAL NET PLAYING IN THE NINTENDO SWITCH**

**Joan Dot Sastre**

**Degree in informatic engineering**

**Faculty of engineering**

**Academic Year 2020-21**

Key words:

word 1, word 2, word 3, …

*Thesis Supervisor’s Name Jose María Buades*

*Tutor’s Name (if applicable) Jose María Buades*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The University is hereby authorized to include this project in its institutional repository for its open consultation and online dissemination, for academic and research purposes only. | Author | | Supervisor | |
| Yes | No | Yes | No |
|  |  |  |  |

**Abstract**

Contents

[1. Introduction 1](#_Toc73975397)

[1.1. Project summary 1](#_Toc73975398)

[1.2. Artificial intelligence in videogames 1](#_Toc73975399)

[1.3 Objectives 2](#_Toc73975400)

[2. Tetris 99 and system build 2](#_Toc73975401)

[2.1 Game basics 3](#_Toc73975402)

[2.2 UI and specifics 5](#_Toc73975403)

[2.2 SRS (Standard rotation system) 7](#_Toc73975404)

[2.3 System 9](#_Toc73975405)

[3. Switch-PC interface 9](#_Toc73975406)

[4. Information capture 10](#_Toc73975407)

[5. Deep learning module 10](#_Toc73975408)

[6. Decision making 10](#_Toc73975409)

[7. Results 10](#_Toc73975410)

[8. Conclusions 10](#_Toc73975411)

1. Introduction

1.1. Project summary

The basis of this project consists in achieving an 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 already shows us that this will not be a trivial matter. The AI has to be built and trained through a PC, and then be able to receive and send information to the console reliably.

1.2. Artificial intelligence in videogames

Artificial intelligence has been present in videogames 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 also currently be found in some games. All these methods are mostly used for NPCs (non-playable characters) 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.

(<https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games>)

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 APIs (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.

Once we have discussed about how data can be collected, we can introduce the next step, agent building. Due to videogames, many different methods have arisen, and with the increasing difficulty of the games beaten has also come an increase in agent intricacy, leading to the drop of simpler techniques in favour of reinforcement learning, which ended up performing much better in highly complex environments.

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 discovering 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 ( <https://openai.com/blog/emergent-tool-use/> ).

1.3 Objectives

The overall goal of the project and how it can be achieved 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, though the challenge trying to be taken has a few major and minor hindrances.

First of all, as a minor inconvenience, the Tetris version we are building our AI on features de SRS (Standard Rotation System) \*, which is a modern rotation system with some unconventional situational rotations. No implementation that can be used has been found so 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.

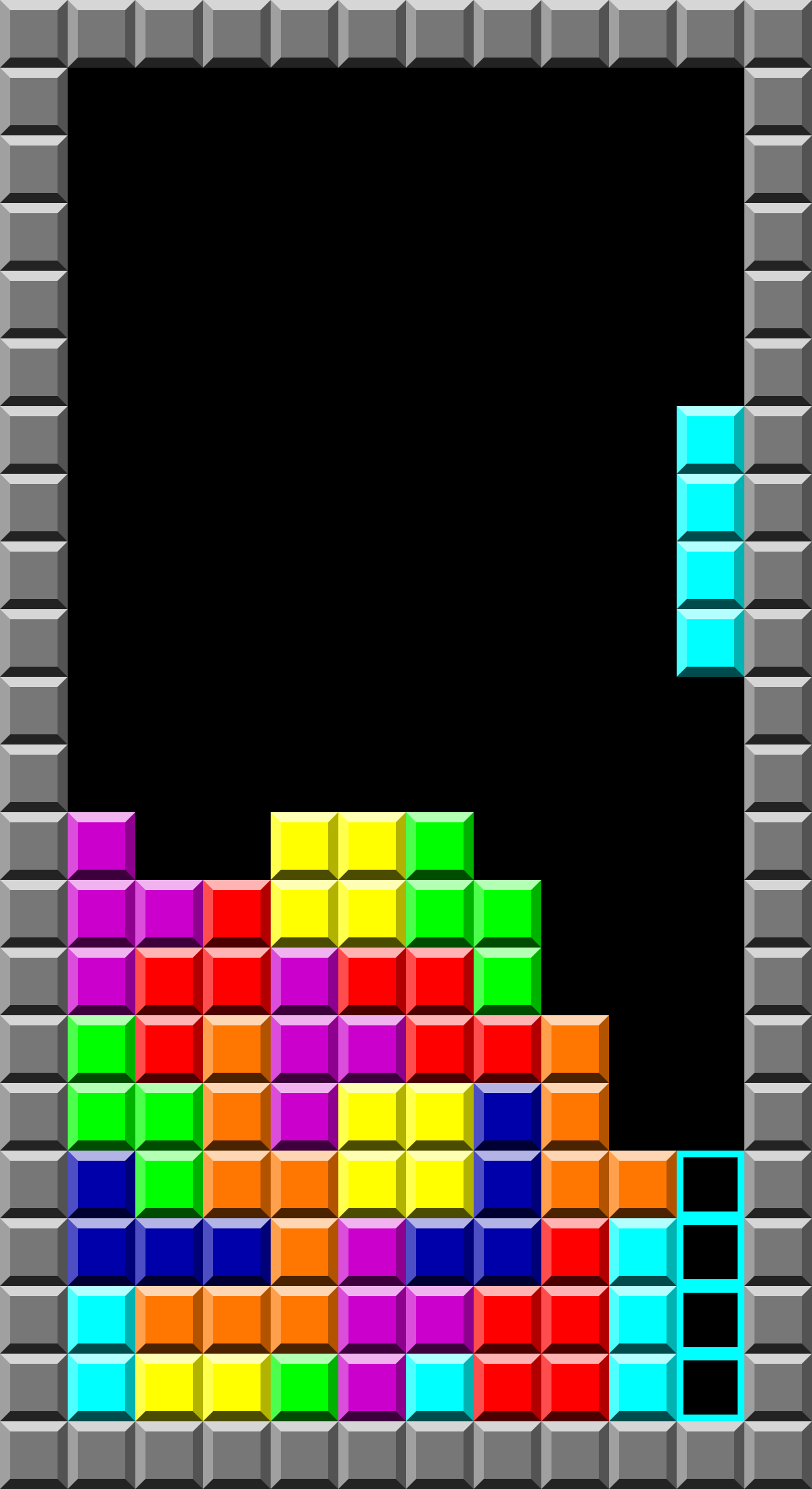
Taking this into account, we expect our agent to work swiftly and, if we manage to pass the neural net’s output accurately to the console, be able to perform well when under low gravity settings.

2. Tetris 99 and system build

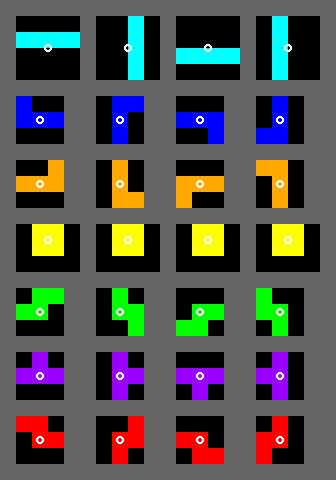
Tetris is a long running game series that has been ongoing since 1984, when Alexey Pajitnov invented it. Ever since it was created, many iterations of the game have been made, with each one of those somewhat altering the rules or adding new mechanics to spice things up. As previously mentioned, the project is being made under the Tetris 99 version, which implements the SRS.

2.1 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 this:



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 and they look as follows:



As we can see in the image above, 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.

The actual formula which dictates how many points we get is:

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

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):

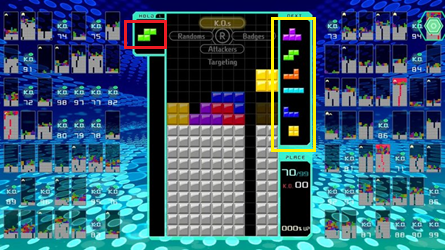
|  |  |
| --- | --- |
| Combo | 50 × combo count × level |
| [Soft drop](https://tetris.wiki/Drop#Soft_drop) | 1 per cell |
| [Hard drop](https://tetris.wiki/Drop#Hard_drop) | 2 per cell |

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.

2.2 UI and specifics

An actual game of Tetris 99 will look like this:



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.

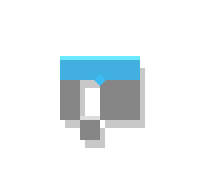
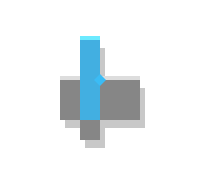
It should also be noted that when receiving garbage lines, those will first be shown in the column right under your piece storage, and only be added to your board after some time. The time is indicated by 3 colour stages, being grey, yellow, and red, from best to worst. Garbage lines can also be cleared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

2.2 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.

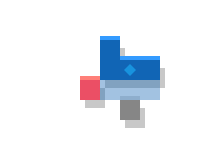
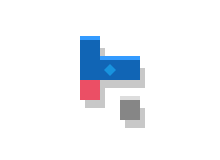
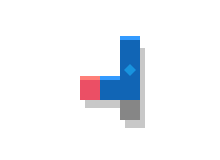
As there is a very large variety of rotations and kicks that can be performed, only a few examples that represent most cases will be shown:

No kick

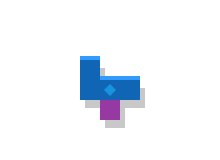
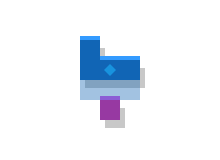
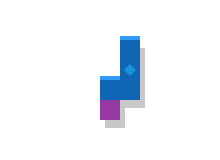
Piece only phases through blocks.

Right kick

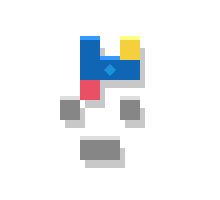
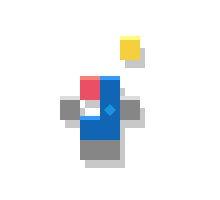
The piece is pushed to the right to rotate.

Up right kick

Piece gets pushed up and right.

Down kick and phase through

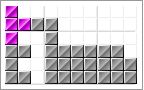
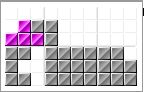
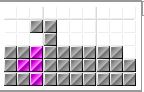
Pieces moves down and phases through red block.

Now that we have seen 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.

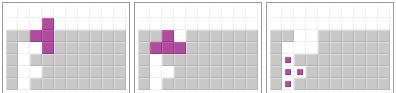
T-spin single



T-spin double

T-spin triple



https://harddrop.com/

2.3 System

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 consists 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 was 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.

3. Switch-PC interface

Aquí hay que explicar el emulador del mando hecho con Arduino, recibe comandos del ordenador y se los pasa a la Switch

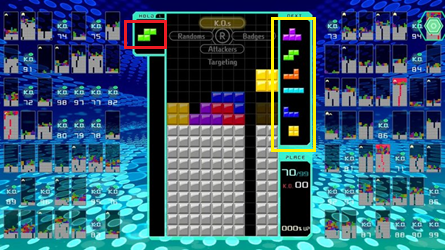
Aunque no lo hayas hecho se puede describir su funcionamiento.

4. Information capture

4.1 Detection

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. To achieve this, we used a normal capture card, so that we can process the input on the pc using OpenCV.

As previously mentioned, a game of Tetris99 looks like the following image:



Tetris game distribution

Given that the only thing which is going to give us the information we need are the pieces, a method to detect a whether there is one and which type it is had to be devised. At first it was thought that the type 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. Thus, came the method to detect pieces, which worked by calculating the mean colour of each shape (only a block was needed) in a 5x5 matrix from its center point, including empty blocks and grey pieces. Still, as seen in the image, when a piece is placed, it turns a darker shade of its former colour, making us have to factor this 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 added when checking for matches, due to other elements in the game board influencing the colour of the own piece with shines or shades.

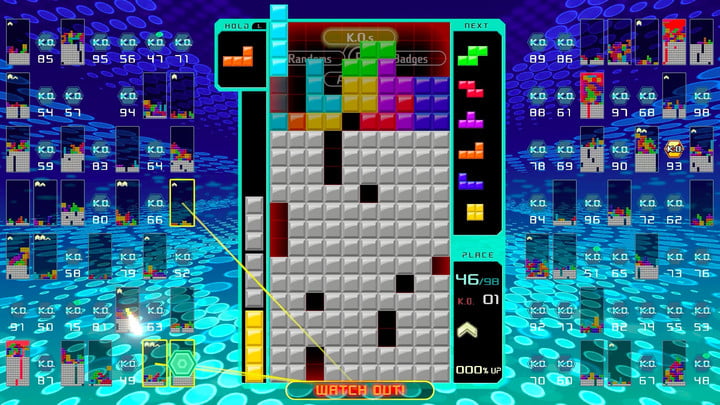
The information we get is also presented back to the user in real time by drawing the conclusions on top of the processed frame. This helps us understand what is being detected and therefore being fed to the neural net.

4.1.1 Game grid

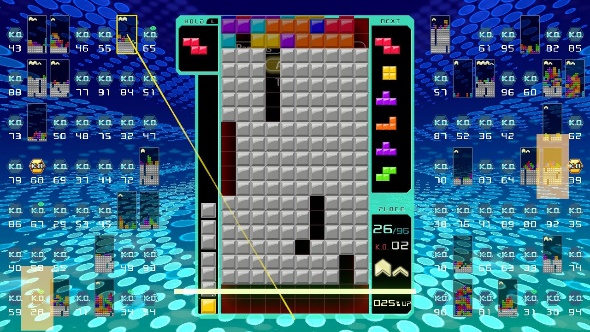
The first element we try to detect is the game grid. It can be done thanks to having manually found where the cells center pixel is, and how wide and tall each cell is. By applying a for loop, we can then iterate through each cell and store the information in two arrays with the size of the game grid (10x20). The first array contains 0s for empty cells, 1s for blocks and 2s 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.



Main piece can be seen



Top placed blocks are invisible

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 build. Each one of those points corresponds to a possible place of 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).

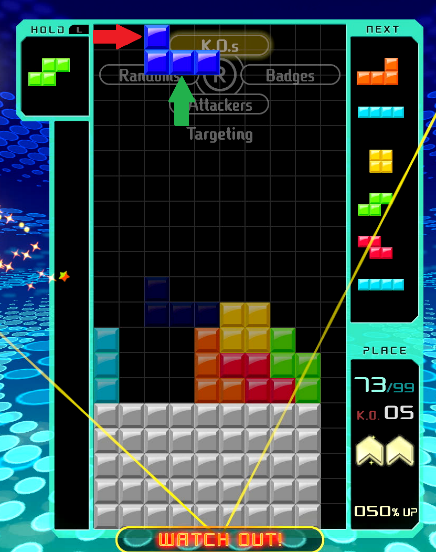
Even though this system was built, it is not currently being used due to it apparently not increasing the neural net’s performance in any way.

4.2 Information adaptation to 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 three rows of the grid, which are the only rows a piece can spawn in.

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 just by checking the first block’s colour but the position is trickier. As all pieces always spawn in the same exact rotation, given a colour we can infer where the center block of the piece, the one giving it its true location, is:



The red arrow shows the first detected block, and the green one is where its true location is (x: +1, y: -1).

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 10x20 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.

5. Deep learning module

Aquí hablar del juego Tetris implementado, y que la red aprende sobre ello

6. Decision making

Explicar que este módulo recoge la información del módulo de captura de información, la procesa con el sistema generado en el

módulo de aprendizaje, encola las órdendes, y las manda emulador del mando

7. Results

Pequeña estadística de partidas jugadas y resultados obtenidos

8. Conclusions