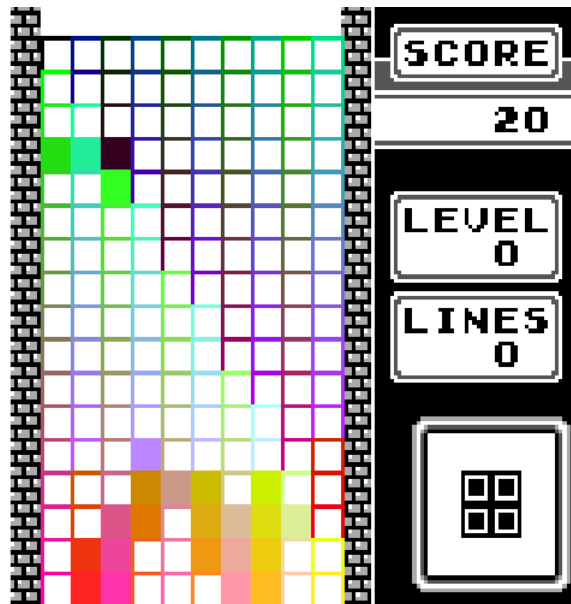


Tetris Playing AI

Machine Learning in Artificial Intelligence



Annaert Yann

Graduation work 2021-2022

Digital Arts and Entertainment

Howest.be

CONTENTS

ABSTRACT	3
Introduction.....	4
Research	4
1. Tetris.....	4
1.1 The Game	4
1.2 History.....	5
1.3 Game Boy Tetris	5
1.4 Techniques And Strategies.....	6
2. C++.....	7
2.1 What Is C++	7
2.2 Why Use C++	7
3. Emulation	7
4. Neural Networks	8
4.1 What Are Neural Networks.....	8
4.2 Neurons in Neural Networks.....	8
4.3 Feed-Forward Neural Networks.....	9
4.4 Training Neural Networks	10
5. Machine learning.....	11
5.1 MACHINE LEARNING Concept.....	11
5.2 TYPES OF MACHINE LEARNING	11
5.3 HEURISTICS.....	12
Case Study	13
1. Data Gathering	13
1.1 PLAYFIELD DETECTION	13
1.2 DATA SQUARES	14
2. ADAPTABLE FEED FORWARD NEURAL NETWORK	15
3. TETRIS AGENT.....	17
4. MACHINE LEARNINGING TOOLS	17
4.1 Data collector.....	17
4.2 Data enhancer.....	18
4.3 Q-table	18
4.4 Simulator.....	19
4.5 heuristics.....	19

5. MACHINE LEARNING EXPERIMENTS.....	20
5.1 CONTEXT	20
5.2 SUPERVISED LEARNING	20
5.3 Q-LEARNING.....	20
5.4 GENETIC ALGORITHM	21
Results	23
Conclusion	24
Reflection and future work.....	25
Bibliography.....	26

ABSTRACT

This paper will cover the creation of an artificial intelligence to play the game of Tetris using machine learning. A Game Boy emulator will be used to play the game of Tetris. This emulator is written in C++ in visual studio. The AI will also be written in the same visual studio project in C++. Finishing the game of Game Boy Tetris with 100 000 points triggers an easter egg. This will be seen as the ultimate goal of the AI.

This paper will cover multiple machine learning techniques as well as taking a closer look at the game of Tetris itself. The decision-making process of the AI uses a neural network to estimate best possible board states. Because of this, the creation of neural networks and how they work will also be covered.

INTRODUCTION

Tetris is and has always been a popular game. To play the game of Tetris, one must make difficult decisions that have a huge impact on the rest of the playtime. The speed at which these decisions must be made increases the longer the game goes on. The best players in the world reach amazing high scores because they know exactly what to do in each situation and have the speed to execute it. But humans are still humans and eventually, humans make mistakes. And the game concludes. But what if we take away the human aspect in this equation? What if we give the controller to an artificial intelligence with the reaction time of a computer?

The goal of this paper will be to create the described artificial intelligence and try to beat humankind (or at least myself) at Tetris. To try and achieve this goal, I ask the question:

In what way can machine learning be used in the process of placing a tetromino in the game of Tetris?

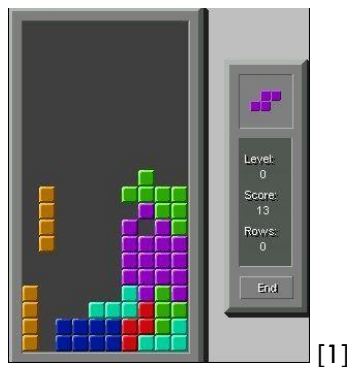
Artificial intelligence is a broad concept. One could write a simple algorithm to play Tetris and call it artificial intelligence. But for the purpose of this paper, I will specifically focus on the concept of machine learning and neural networks as these are topics that lie heavily in my own interests. The most promising machine learning technique in my case was deemed genetic algorithm. This paper will cover all these topics in more detail.

RESEARCH

1. TETRIS

1.1 THE GAME

At its core, Tetris is actually very simple. Blocks fall from the top of the screen. These blocks are called tetrominoes. You as a player have the option to move these blocks around to the left or to the right. You can also rotate them if you so desire. These tetrominoes fall at a specific rate. Once they reach the bottom of the screen they stop and become static. This is when a new tetromino spawns at the top of the screen and the cycle continues.



Author/Copyright holder: @joefoodie. Copyright terms and licence: CC BY 2.0

The goal is to form perfectly filled lines with these blocks. Once a line is formed, the line vanishes, and you gain points. The blocks above this line fall down. If you think you are starting to get good at the game, you can also make the blocks fall down faster. In most cases, a small window displaying the next block is

showcased on the right of the screen. The game ends when the blocks reach the top of the screen the game ends and you are presented with your score.

1.2 HISTORY

Tetris is a puzzle game. It was created by Alexey Pajitnov in 1984. Tetris was inspired by a game called "Pentominoes". This is a game where you assemble shapes in a box. Alexey instead thought of a different setting and imagined the box as a glass with shapes falling into it. Players could control the shape while it falls. The game was an instant hit with the coworkers of Alexey and spread across Moscow.

[2]

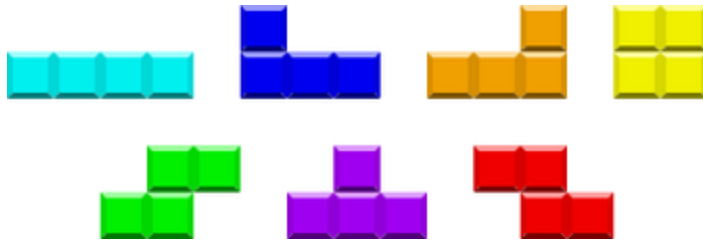
When Pajitnov sent a copy to a colleague in Hungary, it ended up on display in a software exhibit at the Hungarian Institute of Technology, where it came to the attention of Robert Stein, owner of Andromeda Software Ltd., who was visiting the exhibit from the United Kingdom. "Tetris" intrigued Stein. He tracked down Pajitnov in Moscow, but ultimately the game's fate lay in the hands of a new Soviet agency, Elektronorgtechnika (Elorg), created to oversee foreign distribution of Soviet-made software. Elorg licensed the game to Stein, who then licensed it to distributors in the U.S. and the U.K. — Spectrum HoloByte and Mirrorsoft Ltd — The New York Times reported in 1988. According to the Times, "Tetris" was the first software created in the Soviet Union to be sold in America.

1.3 GAME BOY TETRIS

Game Boy Tetris is among the most played Tetris games of all time. It was originally packaged together with the Game Boy system. Because of this, many Game Boy players played Tetris. It was also the first Tetris game to feature 2-player battles.

There are some important details here to consider specifically because our AI will play the Game Boy version of Tetris:

- The Game Boy system runs at 59.73 frames per second.
- Game Boy Tetris features multiple game modes. The mode we are interested in is called A-TYPE. This is a marathon mode where the player can play as long as he can. There are 20 levels in this mode. The current level slowly increases when the player clears lines. Each level increases the speed at which blocks fall.
- The playfield dimensions are 10*18.
- The next piece falling from the top is displayed at the bottom right of the screen. There is only one next piece displayed at a time.
- The player does not have the option to store pieces for future use. This is a feature used in later versions of the game.

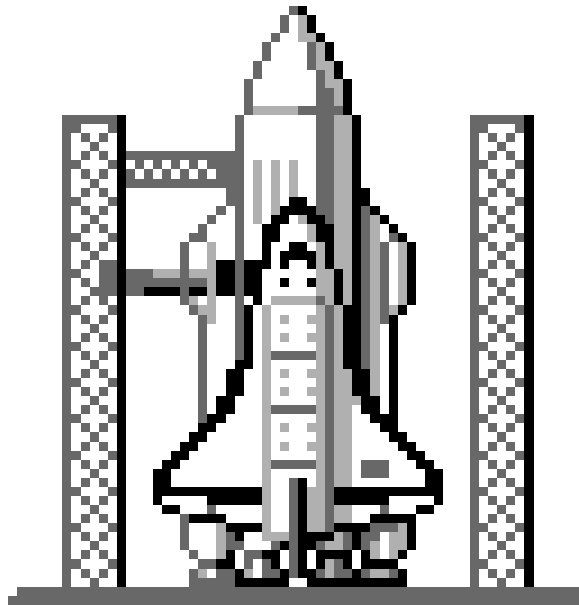


[3]

-There are 7 forms of Tetraminoes possible. These are polyominoes made of four square blocks. I, O, T, S, Z, J and L.

1.4 TECHNIQUES AND STRATEGIES

Cheating may be a valid strategy to achieve the highest scores. When you start the game with the down arrow pressed, you will be able to start ten levels ahead of the level you selected. Moving pieces as fast as you can back and forth will result in the game giving you more of these pieces. This can be abused to get more of a certain desirable piece such as the I shape.



[6]

Finishing the game with 100 000 points or more will set off an animation of a rocket blasting off. This easter egg is often seen as beating the game.

Clearing more lines at once will give you more points. Because of this, it is more desirable to stack pieces together and clear them all at once.

Even though a piece might already have reached the bottom, it is still possible to move that piece. Good players use this to fit a piece in the correct place at the last second.

2. C++

2.1 WHAT IS C++

C++ is a general-purpose, object-oriented programming language created by Bjarne Stroustrup in 1980. It's very similar to its predecessor C and C with classes. C with classes also being the work of Bjarne Stroustrup. "C++ is so compatible with C that it will probably compile over 99% of C programs "[4]. C was developed for programming operating systems, but C++ is more of a general-purpose language. It's often named the "Swiss Pocket Knife of Languages".

2.2 WHY USE C++

C++ may not be the easiest language to work with, yet there are a lot of people still using it for machine learning and artificial intelligence. It is a high-performant language. C++ also has a lot of machine learning libraries.

[5]

C++ is a common choice in machine learning when complex or custom functionality is required. Another strength of C++ is high performance; it is one of the few programming languages in which petaFLOPS computations have been achieved, others being C, Julia, and Fortran. There are many companies that use C++ libraries for machine learning, deep learning, automation, and many other tasks. Some of the popular companies include Airbus, Philips, Microsoft, Thales, Seat, Google, Acciona, and many others.

Because I will use a Game Boy emulator writing in C++, it will be relatively simple to integrate the AI with the emulator if the AI is also writing in the same language in the same project.

3. EMULATION

An emulator enables a computer to act as if it is another computer. In most cases this is a software program that can be run on the so called "host". This program can be used to run other programs on the "guest" computer that is being emulated. One popular example is HP laser printers. A lot of printer software is made for HP laser printers. An emulator can be very cost efficient for other laser printers' brands as they don't have to make their own software in this way. There also exists a large video game console emulator library online. Fans of arcade or older games use these to emulate their favorite games on their original console.

In the Game Boy emulator project we are using, a window is being rendered on the screen using SDL and ImGui displaying the Game Boy in action. We can use the easily accessible pixel buffer generated each frame as a source of information about what is going on inside of the game. This pixel buffer is a simple array of size 160 * 144 containing the color of each pixel. This size is no coincidence as the screen resolution of the original Game Boy is also 160 * 144 pixels.

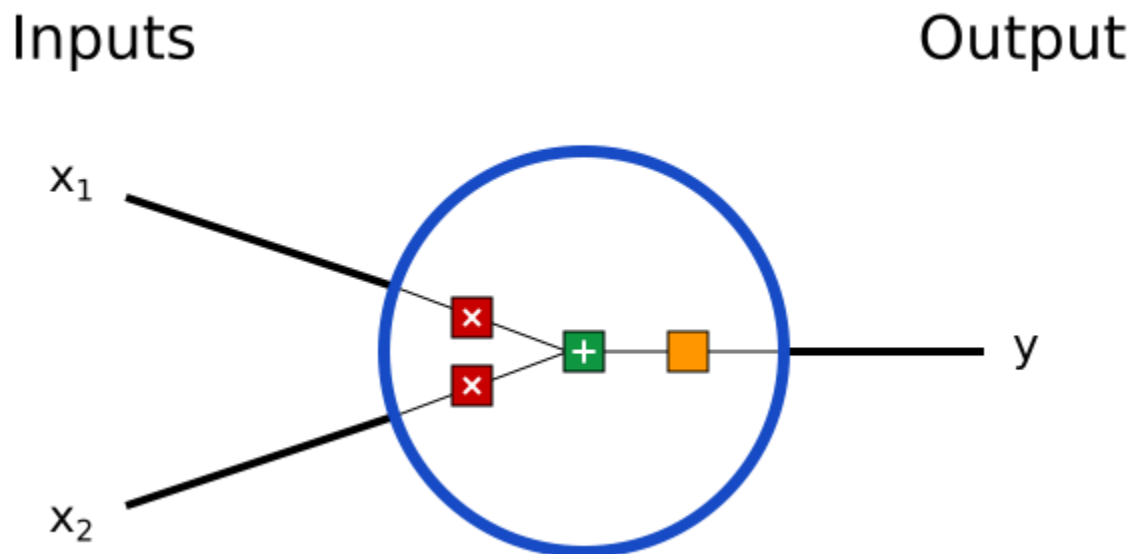
4. NEURAL NETWORKS

4.1 WHAT ARE NEURAL NETWORKS

“Neural networks are machine learning tools built to simulate biological neurons” [7]. Neural networks always consist out of an input and output layers and sometimes hidden layers as explained in [10]. Inside of these layers exist neurons. They are connected through weighted connections. Feeding input through the input layer generates a flow through the network resulting in a or multiple output values in the output layer. These outputs are seen as the prediction of the neural net. The weights of the connections can be changed to gain more desirable predictions based on expected results.

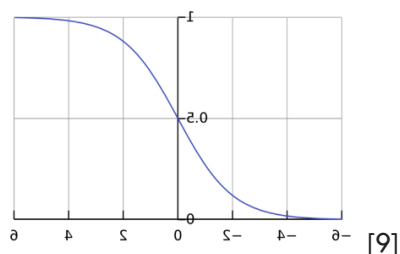
4.2 NEURONS IN NEURAL NETWORKS

Neurons are the basic units of a neural network. “A neuron takes input, does some math with them, and produces one output” [9].



[9]

The image above shows a 2-input neuron. When these inputs reach the neuron, they are multiplied by a weight and added together. After this the sum goes through an activation function. The activation function being a function that transform the output in a predictable form (example: $(-\infty, +\infty)$ to $(0,1)$).

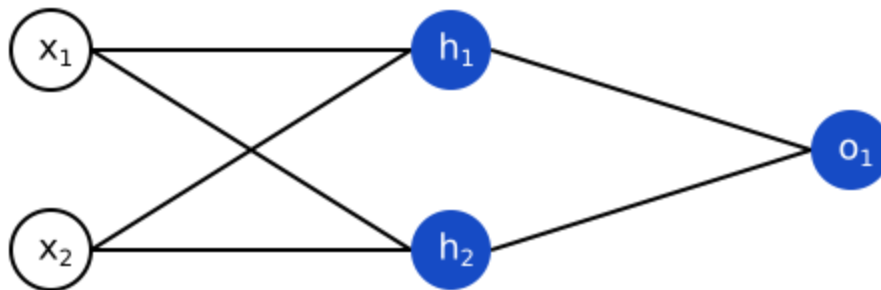


[9]

Input Layer

Hidden Layer

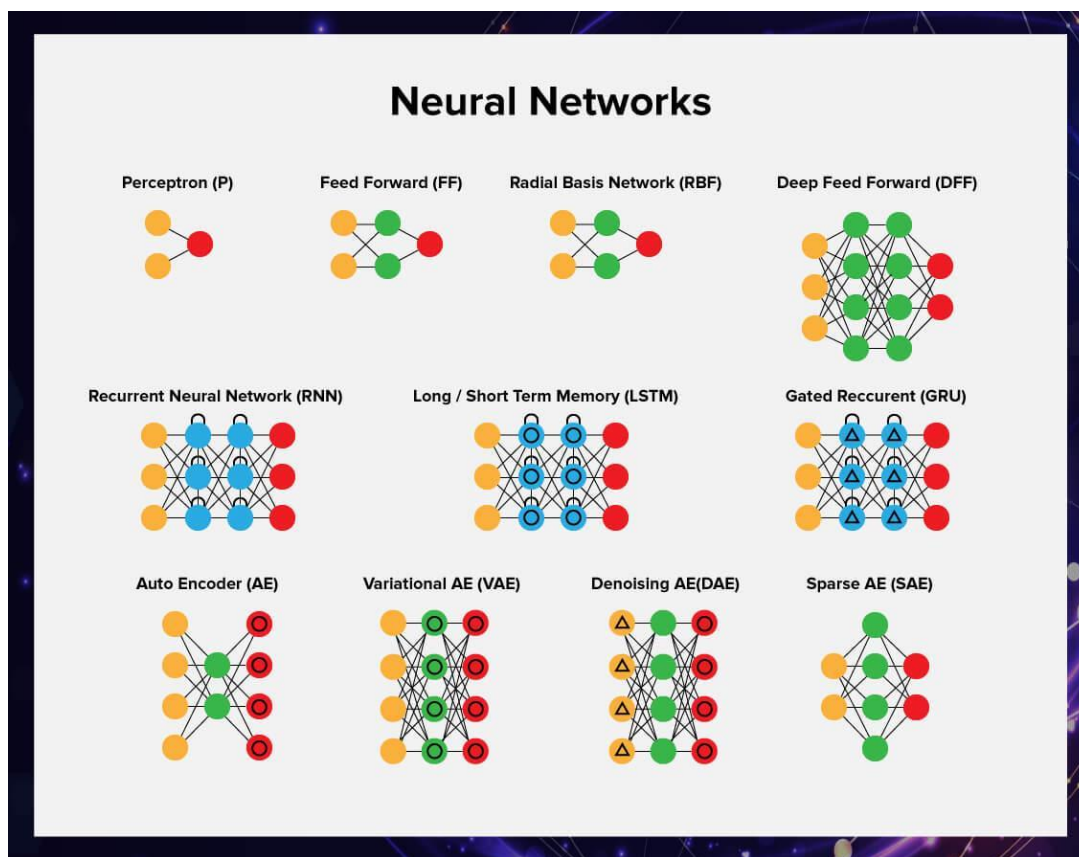
Output Layer



[9]

Connecting these neurons together results in a neural network. The outputs from previous layers are used as inputs for the next layer, thus creating a network. Neural networks always have an input and output layer and sometimes contain a hidden layer. There can be as much hidden layers as needed. Layers can also be as big as needed. We can train these networks to predict results based on the inputs.

4.3 FEED-FORWARD NEURAL NETWORKS



[8]

There are many different neural networks. One of the simplest neural networks is seen as the perceptron. This is a type of feed-forward network. In our case, deep feed-forward networks are the most useful. These are also relatively simple and don't have memory as shown in some examples above. In these types of networks, the data flows in one direction. There are no loops or cycles. There can be multiple hidden layers but it's not a necessity to have any. When there are multiple hidden layers, we can speak of a "deep" neural network.

We can use deep neural networks by first initializing the neurons with random weights, filling in the input values in the neurons in the input layer and letting the data flow through the network. This is called forward propagation as explained in [11]. When we know the desired results corresponding with the inputs given, we can change the weights of the network to minimize its loss.

4.4 TRAINING NEURAL NETWORKS

4.4.1 LOSS FUNCTION

[12]

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the machine learning model. From the loss function, we can derive the gradients which are used to update the weights. The average over all losses constitutes the cost.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_{\text{true}} - y_{\text{pred}})^2$$

[12]

4.4.2 BACKPROPAGATION

[13]

Backpropagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter. Similarly here we also use gradient descent algorithm using Backpropagation.

For a single training example, Backpropagation algorithm calculates the gradient of the error function. Backpropagation can be written as a function of the neural network. Backpropagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.

The main features of Backpropagation are the iterative, recursive and efficient method through which it calculates the updated weight to improve the network until it is not able to perform the task for which it is being trained. Derivatives of the activation function to be known at network design time is required to Backpropagation.

5. MACHINE LEARNING

5.1 MACHINE LEARNING CONCEPT

Machine learning is seen as a type of artificial intelligence. It allows software to “learn” and predict outcomes. This is useful when problems don’t have an immediate answer that can be written in the program. Data is fed through a machine learning algorithm as input and is used to predict desirable outcomes. The algorithm can change itself in a way that allows itself to better reflect “correct” predictions based on feedback or answers given by the program or user.

“Machine learning is important because it gives enterprises a view of trends in customer behavior and business operational patterns, as well as supports the development of new products” [14]. It’s also used in many more fields such as medicine, speed recognition... In our case however it’s seen as a tool to help find solutions to problems that are not immediately obvious or are too difficult to write ourselves.

There are different categories in the field of machine learning. These are based on how the algorithm learns and improves itself. We use different types of machine learning depending on the data we want to predict.

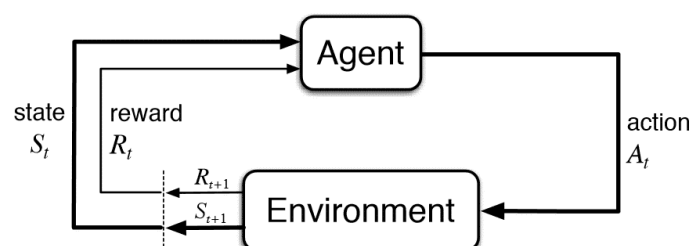
5.2 TYPES OF MACHINE LEARNING

5.2.1 SUPERVISED LEARNING

In supervised learning the user trains the machine learning algorithm himself. To do this we need access to input as well as the corresponding outputs. This set of data is called the training data. The algorithm uses this training data to optimize itself to predict similar outputs with new inputs.” As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately” [15]. A well-trained algorithm will return the correct outputs given inputs from the training data and also predict correct outputs given inputs not in training data.

5.2.2 Q-LEARNING

Q-learning is type of reinforcement learning algorithm. Reinforcement learning trains a model to a solution with no need of training data [16]. There are a few requirements to do this. The agent using predictions from the model needs to be able to interact with an environment. It needs a goal in this environment. Performing actions in this environment needs to be given a reward. These rewards are used to train the model to steer it towards the goal. Rewards can be positive or negative. A simple example is race car being driven on a racetrack. Driving backwards results in a negative reward.

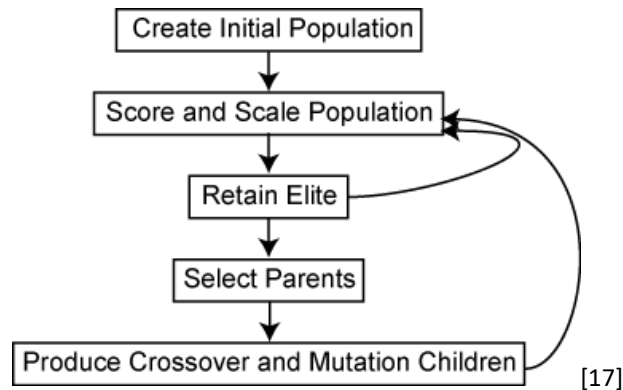


[18]

In Q-learning the best action for an environment state is searched. The agents learn through trial and error. It may find strategies or rules on its own. Because of this Q-learning is seen as model free, off-policy.

To search and find the best action for each state the model keeps a Q-Table. In this table states-action pairs and their corresponding reward are saved. These are called Q-values. Through trial and error, the Q-values are changed with the reward given to a state-action pair.

5.2.3 GENETIC ALGORITHM



In machine learning a genetic algorithm is a method based on natural selection. A population of agents each with their own prediction models is created. These agents can interact with the given environment just as in 5.2.2. Each agent is simulated till an end condition is met. After this, a score is given to the agent, this is called the fitness of the agent. Each agent must be given a score before proceeding to the next step. We can then rank them based on this score and select the best performing agents. These agents are used to breed (crossover) agents into a new population. Because the new population originates from the best agents from the previous generation (previous population), the new generation will hopefully be better than the previous one. New strategies and solutions can be found with each generation.

A percentage of the decision making in agents is mutated with the creation of the new generation to prevent stagnation in the found strategies. This percentage is called the mutation rate. If each agent in a population is going to the right for example and no mutations are made to go left. Going left will be totally unexplored as a valid strategy. Mutating an agent in our case means changing the neural network's weights randomly. When a new population is created, we simulate each agent again and the process continues until an agent performs as desired.

5.3 HEURISTICS

When classic problem-solving methods are too slow, we can turn to heuristics to solve the problem. These solutions may not be the best solution but fall within an acceptable error range. They are often used together with machine learning algorithms. A heuristic function may for example approximate the worth of going left compared to going right and based on this make the decision.

[19]

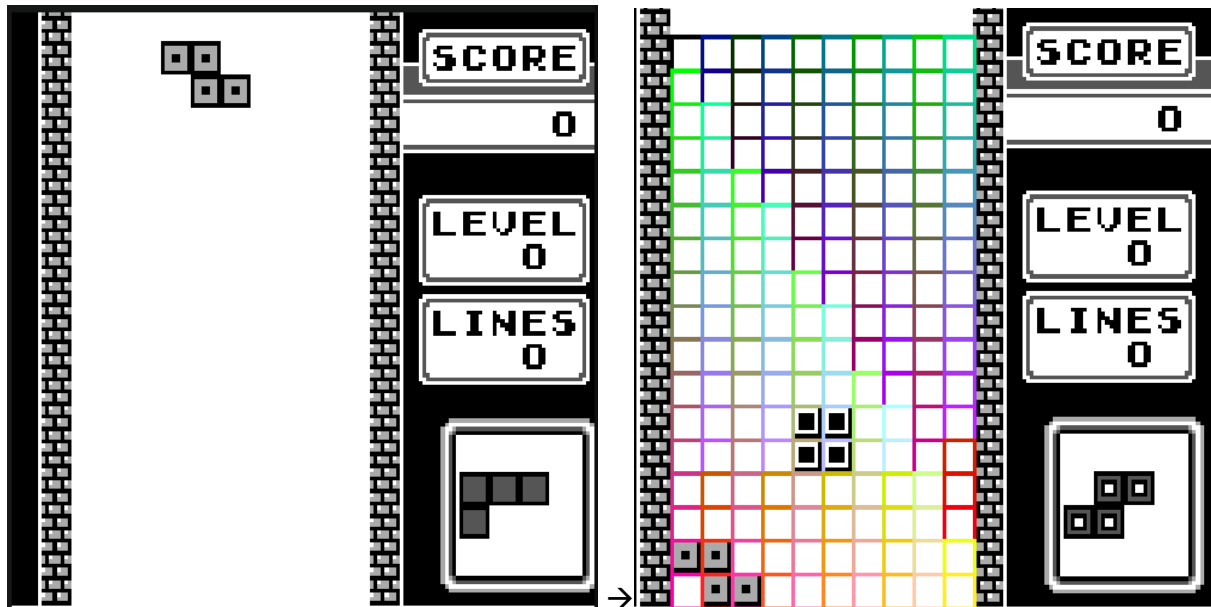
Heuristics are used in machine learning (ML) and artificial intelligence (AI) when it's impractical to solve a particular problem with a step-by-step algorithm. Because a heuristic approach emphasizes speed over accuracy, it is often combined with optimization algorithms to improve results.

CASE STUDY

1. DATA GATHERING

1.1 PLAYFIELD DETECTION

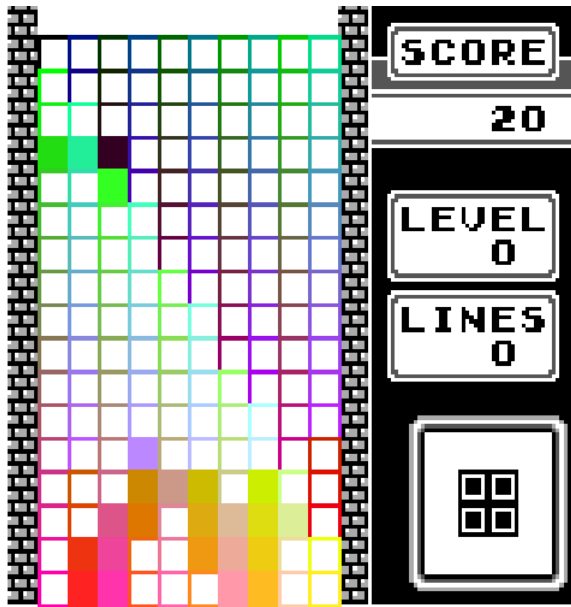
The artificial intelligence will need an environment to interact with, some way of gaining information. The emulator I'm using to play Tetris has its pixel buffer it using to render a window on the screen easily accessible. This buffer contains the colors values.



The first task to get started is making sense of this pixel buffer. In other words, transforming the buffer data into Tetris Data as I call it. Most of the screen space is not useful when you play Tetris. In the left image above, you can see the window being rendered. The bricks at the side of the playfield, the score, the current level, the current lines, it's nice to have these things but they are not essential to play the game. What is important however is the playfield. A playfield in Tetris exists out of a grid of squares. In our case of Game Boy Tetris, the playfield is a grid of size $10 * 18$. With this info we can create and store an array of squares using the correct pixel positions from the pixel buffer. To find the correct square positions I simply used a trial-and-error strategy of coloring pixels till I found the correct one.

After searching for a while, I found that the playfield starts 16 pixels to the right, this being the white line and bricks on the left. Squares have a size of 16 pixels. With this info, the array could be created and rendered to the screen for verification as show in the right picture above. The playfield however has a size of $10*17$ instead of $10 * 18$. This is because the top row is not useful to consider. When blocks reach this height, the game is over anyway.

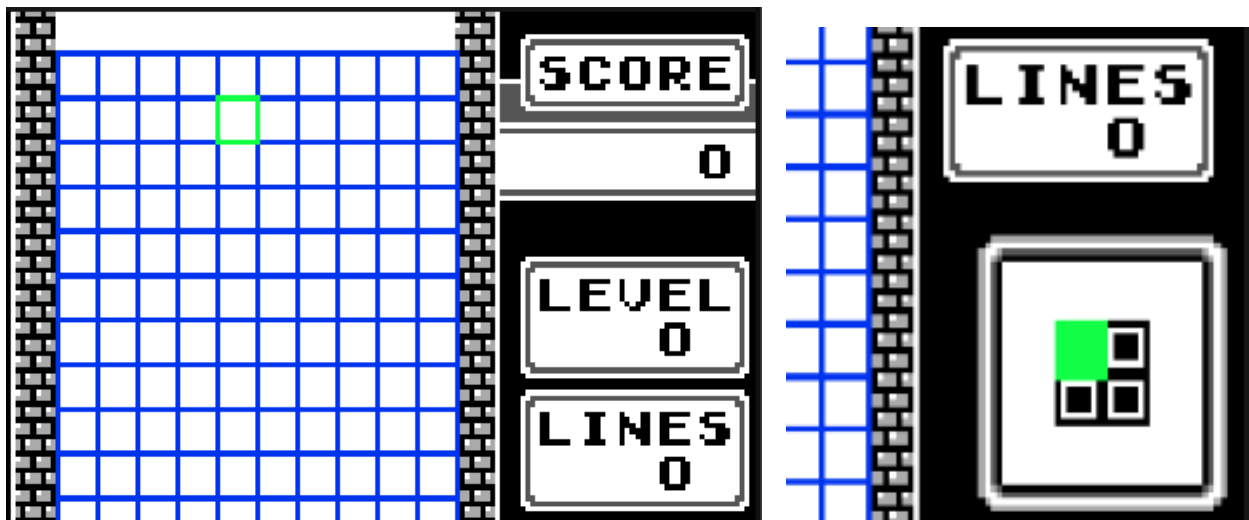
Next up is knowing when a square is being used or not. This is quite simple as all tetrominoes have some black color in them and all the playfield is white. When a square has some black into it that means the square is being used. This information gets stored together with the corresponding square.



With this information we can render the used squares.

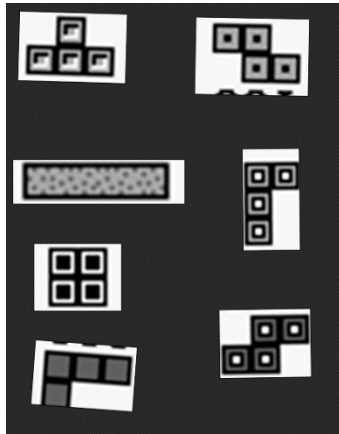
1.2 DATA SQUARES

Knowing what type of tetromino is not important when the pieces already lie in the playfield. It only matters if a square is used or not. It does matter however when a new piece is spawned at the top. Depending on the type of piece, the AI may make different decisions on how to place the piece. To gather this information, I placed a data square as I call them on the spawn position of new pieces.



This data square has the same function of normal squares in the playfield with the extra benefit of tetromino type detection. This information can be used by the AI to know when and how to place a piece. We can now also place a data square on the next piece position to know what type of tetromino the next piece will be and store this information. Notice that the next piece data square is always “being used”.

```
void Enviroment::CheckSquareTetrominoType(const uint16_t(&pixelBuffer)[23040], Data::SquareData& square)const
{
```



Tetromino type detection happens through pattern detection. In more recent versions of Tetris, all tetrominoes have another color. Knowing that there is a light blue pixel inside of the data square means that its type is an I piece for example. In this Game Boy version however the same value of black may be used in different types of tetrominoes. Every piece however has a different pattern that can be found to detect its type.

2. ADAPTABLE FEED FORWARD NEURAL NETWORK

```
class Neuron;
class NeuralNetwork
{
public:
    explicit NeuralNetwork(const std::vector<unsigned>& topology, double learningRate, const double momentum);
    ~NeuralNetwork();

    NeuralNetwork(const NeuralNetwork& other) = delete;
    NeuralNetwork(NeuralNetwork&& other) = delete;
    NeuralNetwork& operator=(const NeuralNetwork& other) = delete;
    NeuralNetwork& operator=(NeuralNetwork&& other) = delete;

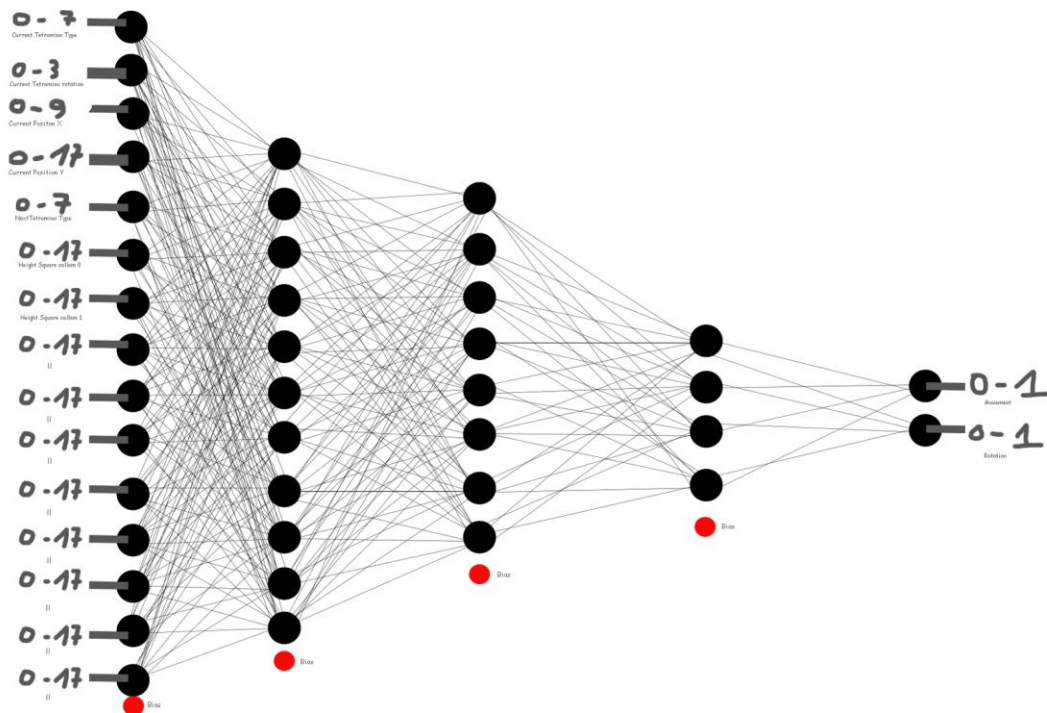
    std::vector<double> Predict(const std::vector<double>& input);
    void Train(const std::vector<double>& input, const std::vector<double>& target);
    double GetAverageError() const { return m_AverageError; }
    std::vector<std::vector<Neuron*>>& GetLayers() { return m_Layers; }

    void Load(const std::string& path, bool crossOver = false);
    void Save(const std::string& path) const;

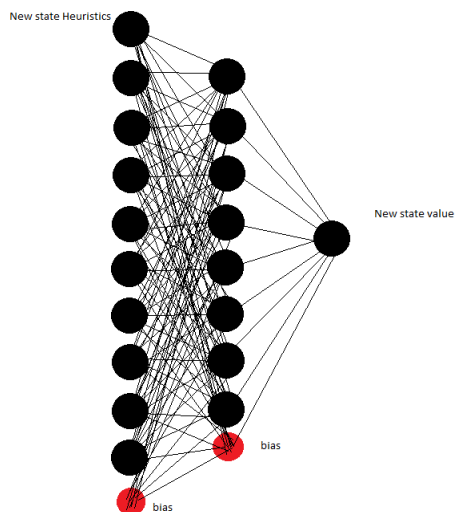
private:
    double m_LearningRate = 0.15;
    double m_Momentum = 0.5;
    double m_Error = 0;
    double m_AverageError = 0;
    std::vector<std::vector<Neuron*>> m_Layers;
    static double SamplesPerAverageError;

    void ForwardPropagation(const std::vector<double>& input);
    void BackPropagation(const std::vector<double>& target);
};
```

I've always been fascinated by the concept of neural networks and wanted to use one in this project. From previous research I found that the neural network had to be a feed forward neural network. Changing the size of the layers and number of layers is also something required as the preferred size is not yet known. To help me achieve this task I found an interesting tutorial [20]. Most of the functions were adapted to fit my own needs and to gain more structure. In the end I ended up a useable and simple class that had all the functionality that was needed.



Throughout the project I used multiple layout types for the neural network. Resulting in some complex variants where all the Tetris data is used as input and a Tetris move as output. A Tetris move being a position and rotation on where to place the piece. First variants used the current Tetris data to generate a go left, go right, and do a rotation output. To train a network to know when to go left or right and or do a rotation on every possible board state in every moment of the piece falling is very difficult. Therefore, it is more useful to let the network decide on where the piece should land and steer the piece to that position automatically. In the end the network used is relatively simple with a 10-8-1 layout depicted below. This neural net uses heuristics on the state of the playfield that a Tetris move would result in. The net then generates a value for the board state deciding how valuable that board state would be. When doing this for every possible move on the current board state, we can decide which move is the most desirable.



3. TETRIS AGENT

```
class TetrisAgent
{
public:
    explicit TetrisAgent(const std::string& path);
    ~TetrisAgent();

    TetrisAgent(const TetrisAgent& other) = delete;
    TetrisAgent(TetrisAgent&& other) = delete;
    TetrisAgent& operator=(const TetrisAgent& other) = delete;
    TetrisAgent& operator=(TetrisAgent&& other) = delete;

    TetrisAction* Update(const Data::TetrisData& data, float deltaTime);
};
```

The Tetris agent is what will interact with the environment previously created. Each time the environment changes, the data is fed to the Tetris agent. It will then decide what to do and return an action. An action can be empty or contain a Tetris move with a position and rotation. When the agent returns an empty action, nothing happens, and the program continues till the environment changes again. When the agent returns a Tetris move, the move must be performed. Performing the move is not done by the agent itself. The agent just decides and returns the most preferable action at that moment. A simple algorithm inside of the program itself then decides what to do with this information and performs the action. To perform a Tetris move, a piece is moved towards the position that was given and rotated to the rotation that was given. To move a piece to a position means to move it to the column position. The agent's decisions are based on the machine learning technique implemented.

4. MACHINE LEARNING TOOLS

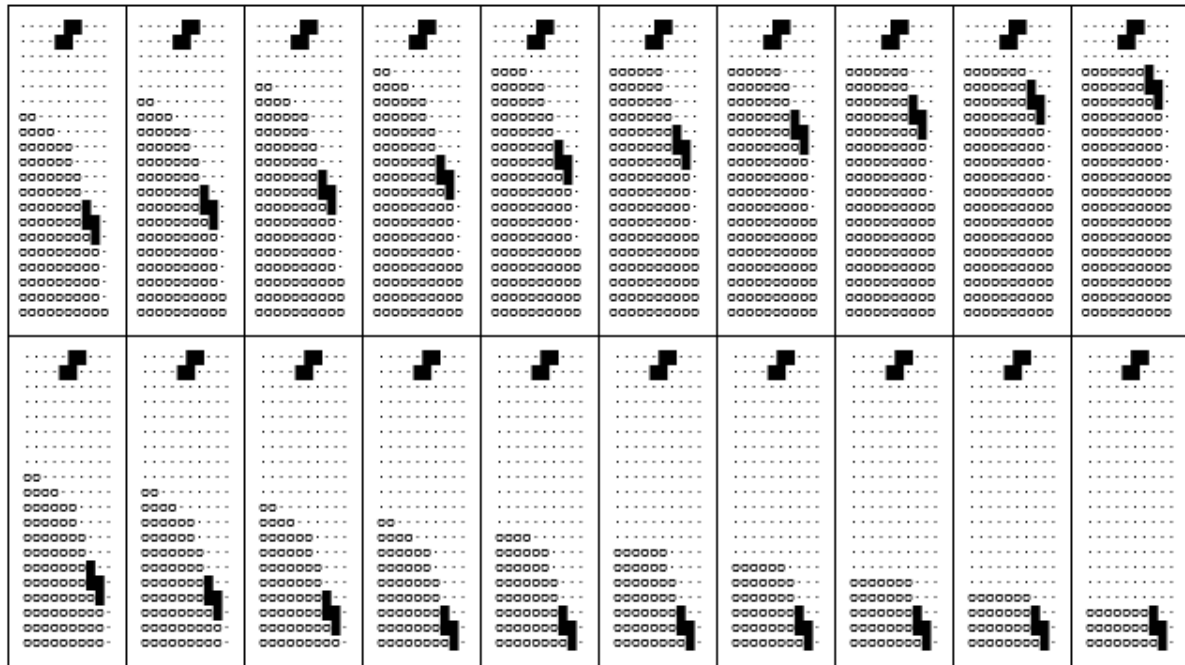
4.1 DATA COLLECTOR

```
0
0
1
1
1
1
1
1
0
0
0
1
1
0
1
1
1
1
0
0
0
1
1
-
0.666667
0
```

Each time the environment changes the agent decides what action to take. Each time a piece spawns the agent will decide to make a move on a certain position and rotation. To try and train a neural network to make this move in a correct way, a neural network needs a lot of data. I decided to collect this data myself to try and make the agent play in a way that reflects me. To do this instead of letting the agent perform a move each time a piece spawns, we collect the position and rotation the player, in this case me, makes together with the board state that move was made on. With this state-action pair data we can train the neural net. The resulting file is a text file containing 0 and 1's. These being the playfield of size 10*17 squares. A 1 means the square is active. And the move that was made on that board state separated by a - sign. This is a position and a rotation indicated with a value between 0 and 1.

4.2 DATA ENHANCER

Most moves in Tetris are very similar when taken a level up or down. This means that we can get a lot of extra data by simply adding more lines to a simulated move. This extra data can twenty-fold the current file size to account for every possible board state that move can be applied to. I got the idea from [21]. Removing all the gaps in the board also takes a way a lot of possible board states. We just must remember to remove the gaps in the data the agent takes as input as well.



[21]

4.3 Q-TABLE

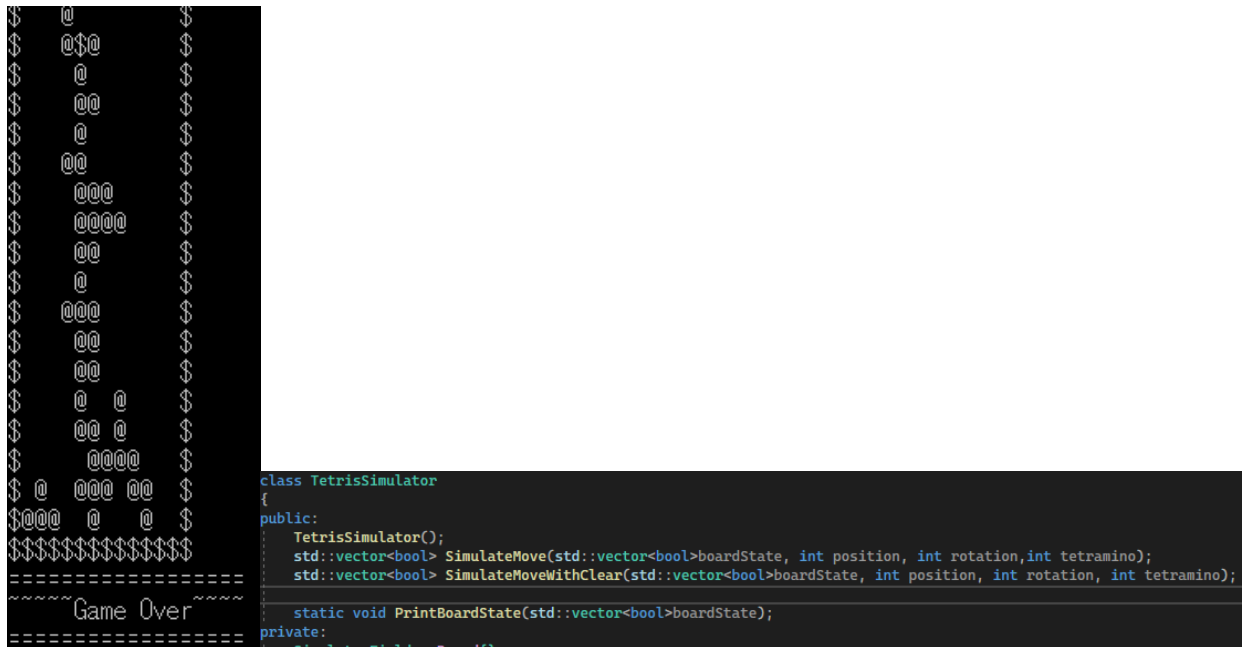
A Q-Table in our case hold the state-action pairs previously discovered. These pairs have a Q-value assigned to them. It is possible to look up the Q-value of a pair by giving the pair as an input. We can change the Q-value of pairs based on rewards. This Q-value decides how “good” a state-action pair is to perform.

```
class QTable
{
public:
    QTable() = default;
    ~QTable();

    bool GetQValue(QAction* action, QState* state, float& value);
};
```

4.4 SIMULATOR

Because the game is played on an emulator, we don't have access to the source code of the game itself. This makes it not possible to know where a piece will land before the piece has already landed there. To fix this issue I created a Tetris simulator that can simulate the resulting board state when given a board state, the piece that will spawn a Tetris move. This simulator is created from a Tetris console game [22] as a base. It uses the logic written in that console game to simulate a certain number of time steps. It has the option to remove lines in the simulation if a line is filled.



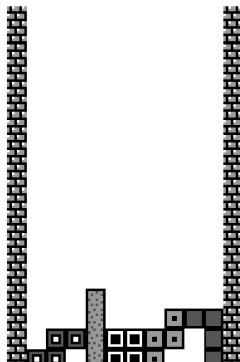
4.5 HEURISTICS

To know how “good” a board state is we can use heuristics to approximate the value of that board state. These can be used as an input in the neural network to generate board values. A great example and one of the most important heuristics is bumpiness. This is the sum of the height differences in each column. Some more examples are: the highest row, aggregate height, number of holes, lines cleared, roughness...

Bumpiness

The sum of absolute height differences between the columns. In the board below,

it's: $1 + 0 + 2 + 2 + 0 + 0 + 1 + 0 + 0 = 7$



[23]

5. MACHINE LEARNING EXPERIMENTS

5.1 CONTEXT

Throughout my experiments I used multiple machine learning techniques. Therefore, tools previously discussed feature tools used in different types of machine learning. Not all experiments were successful but are featured in here anyways as they helped me better understand the concept of machine learning in general and all the possibilities.

5.2 SUPERVISED LEARNING

```
---Move1---  
move to position: 9  
Rotate to rotation: 3  
-----
```

This is the “simplest” version of the AI I created. The agent deciding the moves has a neural network installed into itself helping it predict the next best move. It gains information directly from the environment and uses that information as inputs for the neural network. This information contains the whole board state, the current piece type and the next piece type. Using this information, the neural network predicts 2 outputs values. These are the position and rotation the piece should be moved towards. Moving the piece is done by an algorithm inside of the program itself. The neural network was trained beforehand by data collected and enhanced by the Tetris data collector / enhancer discussed in 4.1 and 4.2. The neural network needs a lot of data before it starts to learn. I played around 30 games of Tetris with data collection activated to collect this data. The neural net then loops several times over this data using backpropagation (4.4.2) to train itself.

5.3 Q-LEARNING

The Q-learning version does not have a neural network installed in the Tetris agent. Instead to decide which move to make at a certain board state, it looks up the best move in a Q-Table (4.3). To decide which move is best it looks up every possible move at the current board state. If the state- action pair is not found, it is added to the table with a value Q-value of 0. The state- action pair with the highest Q-value is chosen as the best one. The move from the state- action pair is then returned by the agent and performed by an algorithm inside of the program itself.

```
-State tetramino  
1  
-State CollumHeights  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
0  
-Action Position  
0  
-Action Rotation  
0  
-QValue  
2  
...
```

Each time a move was performed the agent remembers what move it made that turn. The next turn when a new board state is developed, the agent knows what state-action pair led to that new board state. It can then evaluate how good the new board state is compared to the previous one and give a reward based on the board value difference to the state-action pair that was remembered. This way Q-values are updated and there may be a best new move to perform at a certain board state.

To prevent stagnation in what move is best, the agent performs a percentage of its moves at random. This way more Q-values are updated, and the same move will not be repeated.

5.4 GENETIC ALGORITHM

```
UPDATEGENERATION: Generation: 0 Iteration : 18
Previous Fitness: 6
UPDATEGENERATION: Generation: 0 Iteration : 19
Previous Fitness: 24
UPDATEGENERATION: Generation: 0 Iteration : 20
Previous Fitness: 9
UPDATEGENERATION: Generation: 0 Iteration : 21
Previous Fitness: 7
UPDATEGENERATION: Generation: 0 Iteration : 22
Previous Fitness: 11
UPDATEGENERATION: Generation: 0 Iteration : 23
Previous Fitness: 11
UPDATEGENERATION: Generation: 0 Iteration : 24
Previous Fitness: 8
UPDATEGENERATION: Generation: 0 Iteration : 25
Previous Fitness: 10
UPDATEGENERATION: Generation: 0 Iteration : 26
Previous Fitness: 9
UPDATEGENERATION: Generation: 0 Iteration : 27
Previous Fitness: 9
UPDATEGENERATION: Generation: 0 Iteration : 28
Previous Fitness: 9
UPDATEGENERATION: Generation: 0 Iteration : 29
Previous Fitness: 22
UPDATEGENERATION: Generation: 0 Iteration : 30
Previous Fitness: 24
UPDATEGENERATION: Generation: 0 Iteration : 31
Previous Fitness: 26
UPDATEGENERATION: Generation: 0 Iteration : 32
```

The genetic algorithm AI again uses the neural net to predict the best move at a certain board state. It does not by letting the neural net predict the best move, but by deciding how valuable a new board state is. When we know each possible board state and the value assigned to that board state, we can choose the one with the highest value as the best one. To know each possible next board state, we must first know the current board state. This is given as an input by the environment. We must then perform every possible action on that board state to know what all the next possible board states are. To do this the AI uses the Tetris simulator (4.4). When the best next possible board state is found we can return the move that led to that board state as the best move at that moment. Yet again the algorithm performs the move. There are 10 columns the piece can be placed on. Combine this with 4 possible rotations and we get the number of times the next board state must be simulated. When we account the next piece into this equation, we get

40 * 40 possible board states. This can be quite expensive and lead to some lag. In the following screenshot we can see a board being simulated and the corresponding action being returned.

```
Simulated Move:
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
-----
Action:
ActionValue: -0.27575
Position: 3
Rotation: 1
-----
```

The neural net with a 10-8-1 layout used by the agent decides how valuable a board state is based on heuristics. These heuristics are filled into the input layer and result in the value in the output layer. The heuristics used are [23]:

-The highest row

Counting the bottom of the playfield as 0. Each square upwards means +1. The highest square in the playfield used.

-Aggregate height

The sum of all the row heights in the playfield.

-Number of holes

Each square not used with a used square above it is counted as a hole.

-Columns with holes

If a column in the playfield has one or more holes in it is counted as +1.

-Roughness

The sum of the height differences between columns.

-Row transitions

When squares to the left or right of a used square are empty it is counted as +1.

-Column transitions

When squares to the bottom or up of a used square are empty it is counted as +1.

-Number of pits

Columns without squares.

-lines cleared

Number of rows that are completely filled with squares.

-relative height

Difference of the highest and lowest row.

The neural network is trained using a genetic algorithm (5.2.3) with a population of size 100 and a mutation rate of 10 percent. Each agent is simulated playing the game on the emulator and given a score based on lines cleared. Simulating the game in real time takes a lot of time, especially when the agents start to get good at the game. To counter this issue, I simulated the games using the Tetris simulator (4.4) in a separate program without the emulator. This dramatically sped up the training process.

RESULTS

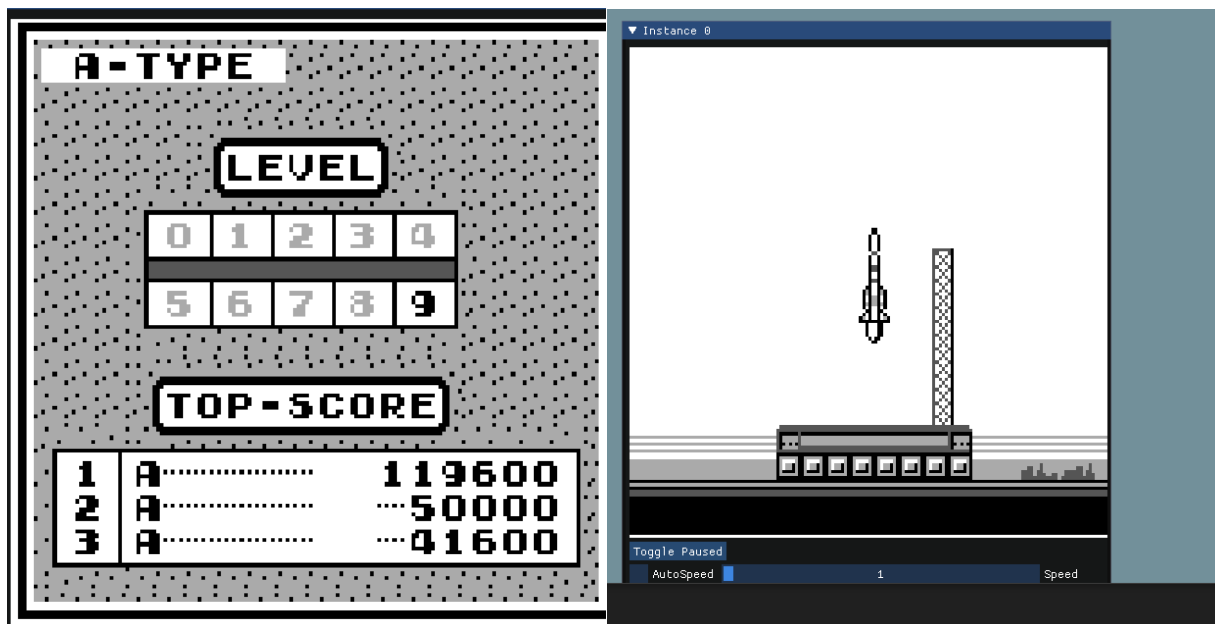
The supervised learning AI had some issues dealing with the data it was presented. Some of the data in the training data file was not useful and even harmful to the neural net. This is because when playing Tetris, myself, I made moves that were impossible for the AI to perform. For example, making a move on the very last second to fill a gap on the left or right. The current AI can only make moves going the correct column all at once.

Trying to make fewer invalid moves in the training data also did not lead to a well performing AI. Even when enhanced the data was just enough. A lot more games should be played before the training data is useful as Tetris has way too many possible board states to cover in the 30 games, I played myself.

The Q-learning variant suffered from a similar problem. Tetris has too many possible state-action pairs to cover in simple Q-table. The Q-table that was generated after around 10 hours of letting the AI play got so huge that loading it in again took around half an hour. The data it had collected at this point was again, just not enough. It needs way more playtime to cover all the possible state-action pairs. Even this will not be enough because only finding and adjusting the Q-value of every possible state-action pair once is not meaningful.

The genetic algorithm variant had much better results. Implementing the Tetris agent using the board heuristics with random weights already led to the AI playing the game in a dumb but acceptable way. This was before the genetic algorithm was even run. After running the genetic algorithm and letting the AI play the game on the emulator for around 5 hours, the results were already decent. It plays Tetris much

better than I ever will but was never reaching the goal of the rocket easter egg found at 100 000 points. I figured the AI needed way more simulating time as the current best agent resulted from only the 9th generation. When using the Tetris simulator to remove the need for the emulator and speed up the training process, the algorithm reached generation 30 at around 2 hours of simulating. The best new agent plays very decently and eventually reached the goal of 100 000 points after around 10 tries.



CONCLUSION

Throughout this project, multiple machine learning techniques were used. Not all of them turned out to be successful, but all of them had meaningful lessons to be learned.

Both supervised and Q learning were useful to help the AI become better at placing pieces on the board. But both suffered from the same problem, this being size. Tetris is a game where a lot of decisions must be made and to learn all the possible decision one can make, is simply too much. In the case of supervised learning, the data enhancer helps to approximate a lot of these decisions, but a lot of data is still missing. In the case of Q-learning, the size issue is too big to overcome. The AI would have to play for thousands of hours to find every possible state-action pair and update the Q-values in a meaningful way. The size of the data file would be unmanageable.

Heuristics and genetic algorithm were absolutely a great way to help learn the AI to place tetrominoes on the board. This is because the heuristics approximate a lot of decisions that otherwise would have to be learned by the AI itself. Taking away as much of these state-action pairs to remember is key. The genetic algorithm was quick to find the optimal solution for the neural network. This makes it a great choice as the machine learning technique in the case of Tetris. This does not mean however that genetic algorithm is the only way to achieve this goal. There are still a lot of other techniques out there.


REFLECTION AND FUTURE WORK

This project taught me a lot about machine learning, neural networks, and AI in general. It's definitely a field that interests me and something that I want to learn more about in the future. The final AI is much better than I will ever be at Tetris and in my eyes that's a success. I created multiple tools that help with the machine learning process and that can be reused in future projects.

Both the supervised and Q-learning approach need more work to a viable strategy. To make the supervised approach work, I need a new way to gather data. Gathering the data, myself is too inefficient. In the future I might look at gathering data from videos [21]. The Q-learning approach however needs to be reworked into another reinforced learning approach. Looking up state-action pairs in a Q-table is too inefficient. To achieve this in the future I could install the neural network and train it using rewards gathered from resulting board states. Using the simulator, the learning process might be fast enough to be viable. For future portfolio work I will apply my learnings I found here in other game AI's.

BIBLIOGRAPHY

- [1] Kumar, J. M., Herger, M., & Dam, R. F. 2021. A Game explained (an example of a single game and how it meets the rules of fun). The Interaction Design Foundation. Retrieved from <https://www.interaction-design.org/literature/article/a-game-explained-an-example-of-a-single-game-and-how-it-meets-the-rules-of-fun#:~:text=The%20aim%20in%20Tetris%20is,re%20sure%20of%20your%20positioning.>
- [2] Weisberger, M. (2016, October 13). The bizarre history of 'tetris'. LiveScience. Retrieved August 15, 2022, from <https://www.livescience.com/56481-strange-history-of-tetris.html>
- [3] Tetromino. Tetris Wiki. (n.d.). Retrieved August 15, 2022, from <https://tetris.fandom.com/wiki/Tetromino#:~:text=A%20tetromino%2C%20is%20a%20polyomino,has%20used%20Tetrimino%20since%202001.>
- [4] Thompson, B. (2022, July 19). What is C++? basic concepts of C++ programming language. Guru99. Retrieved August 15, 2022, from <https://www.guru99.com/cpp-tutorial.html>
- [5] Can C++ be used for machine learning. Arkiana. (2022, August 8). Retrieved August 15, 2022, from <https://arkiana.com/can-c-be-used-for-machine-learning/>
- [6] Tetris (game boy) - hard drop - tetris wiki. (n.d.). Retrieved August 15, 2022, from [https://harddrop.com/wiki/Tetris_\(Game_Boy\)](https://harddrop.com/wiki/Tetris_(Game_Boy))
- [7] Lundgaard, N., & Mckee, B. (1970, January 1). [pdf] reinforcement learning and neural networks for Tetris: Semantic scholar. [PDF] Reinforcement Learning and Neural Networks for Tetris | Semantic Scholar. Retrieved August 15, 2022, from <https://www.semanticscholar.org/paper/Reinforcement-Learning-and-Neural-Networks-for-Lundgaard-Mckee/a922402882fb621c3f39a72d50779440041a9321>
- [8] Gavrilova, Y. (2020, October 8). A guide to deep learning and neural networks. Serokell Software Development Company. Retrieved August 15, 2022, from <https://serokell.io/blog/deep-learning-and-neural-network-guide>
- [9] Zhou, V. (2019, December 20). Machine learning for beginners: An introduction to neural networks. Medium. Retrieved August 15, 2022, from <https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9>
- [10] Malik, F. (2019, May 18). Neural network layers. Medium. Retrieved August 15, 2022, from <https://medium.com/fintechexplained/neural-network-layers-75e48d71f392#:~:text=There%20will%20always%20be%20an,each%20layer%20has%20its%20purpose.>
- [11] Luhaniwal, V. (2020, October 24). Forward propagation in Neural Networks-simplified math and code version. Medium. Retrieved August 15, 2022, from <https://towardsdatascience.com/forward-propagation-in-neural-networks-simplified-math-and-code-version-bbcfef6f9250>
- [12] Seb, P. by, Author Seb, A., & Posts, R. (2022, April 4). An introduction to neural network loss functions. Programmathically. Retrieved August 15, 2022, from <https://programmathically.com/an-introduction-to-neural-network-loss-functions/#:~:text=The%20loss%20function%20in%20a,all%20losses%20constitutes%20the%20cost.>

- [13] Backpropagation process in deep neural network - javatpoint. [www.javatpoint.com](https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network). (n.d.). Retrieved August 15, 2022, from <https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network>
- [14] Burns, E. (2021, March 30). What is machine learning and why is it important? SearchEnterpriseAI. Retrieved August 16, 2022, from <https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML>
- [15] By: IBM Cloud Education. (2020, August 19). What is supervised learning? IBM. Retrieved August 16, 2022, from <https://www.ibm.com/cloud/learn/supervised-learning#:~:text=Supervised%20learning%2C%20also%20known%20as,data%20or%20predict%20outcomes%20accurately.>
- [16] Simplilearn. (2022, July 25). What is Q-learning: Everything you need to know: Simplilearn. Simplilearn.com. Retrieved August 16, 2022, from <https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning>
- [17] Select a web site. What Is the Genetic Algorithm? - MATLAB & Simulink. (n.d.). Retrieved August 16, 2022, from <https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html#:~:text=The%20genetic%20algorithm%20is%20a,a%20population%20of%20individual%20solutions.>
- [18] Ossenkopp, P., sakethv321, Burmeister, C., Mayer, M. H. & J., Marco Radic & Dr. Harald Bosch, & Vallejo, K. W. & G. (2020, February 13). Reinforcement learning – part 1: Introduction to Q-learning. Novatec. Retrieved August 16, 2022, from <https://www.novatec-gmbh.de/en/blog/introduction-to-q-learning/>
- [19] Techopedia. (2022, March 3). What is heuristic? - definition from Techopedia. Techopedia.com. Retrieved August 16, 2022, from <https://www.techopedia.com/definition/5436/heuristic#:~:text=Heuristics%20are%20used%20in%20machine,optimization%20algorithms%20to%20improve%20results.>
- [20] YouTube. (2018, September 11). Neural net implementation in C++. YouTube. Retrieved August 15, 2022, from <https://www.youtube.com/watch?v=sK9AbJ4P8ao>
- [21] Srdjan. (2020, October 16). Machine learning: Ai plays Tetris with Convolutional Neural Network. Ask For Game Task. Retrieved August 17, 2022, from <https://www.askforgametask.com/tutorial/machine-learning/ai-plays-tetris-with-cnn/>
- [22] Kimth. (n.d.). KIMTTH/console-tetris:  most simplest console based Tetris source code by C++. GitHub. Retrieved August 17, 2022, from <https://github.com/kimth/console-tetris>
- [23] Bui, D. A. (2020, June 11). Beating the world record in Tetris (GB) with genetics algorithm. Medium. Retrieved August 17, 2022, from <https://towardsdatascience.com/beating-the-world-record-in-tetris-gb-with-genetics-algorithm-6c0b2f5ace9b>