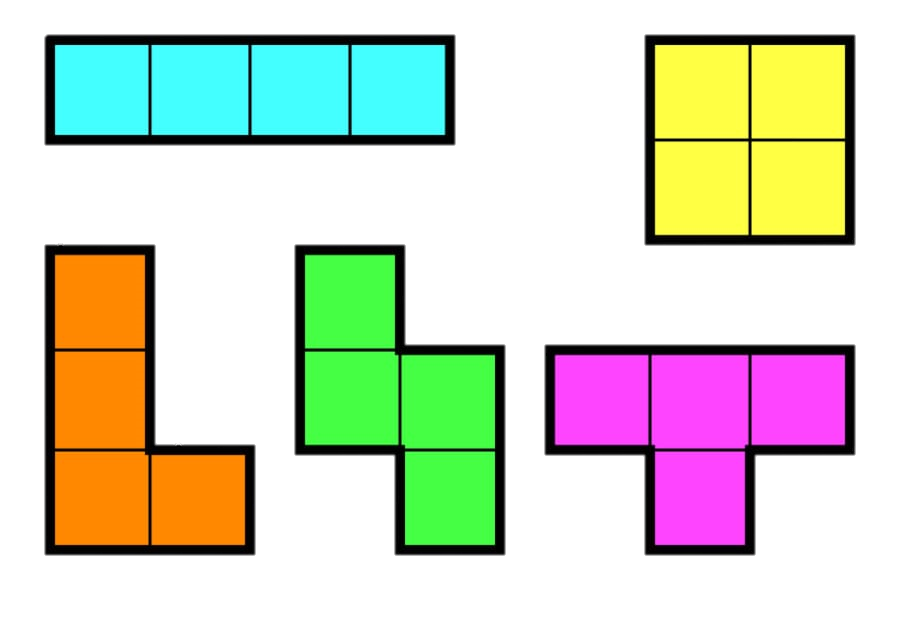
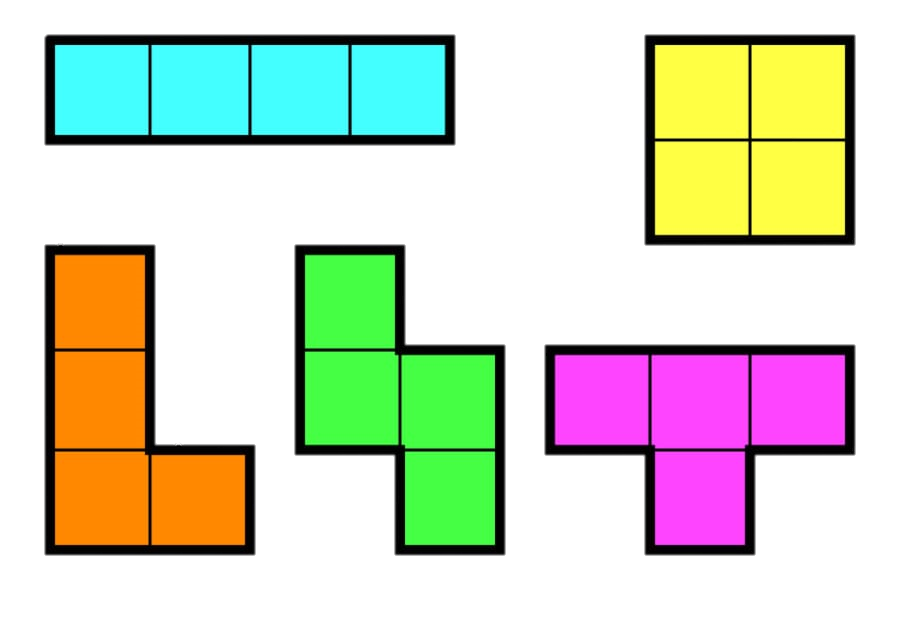
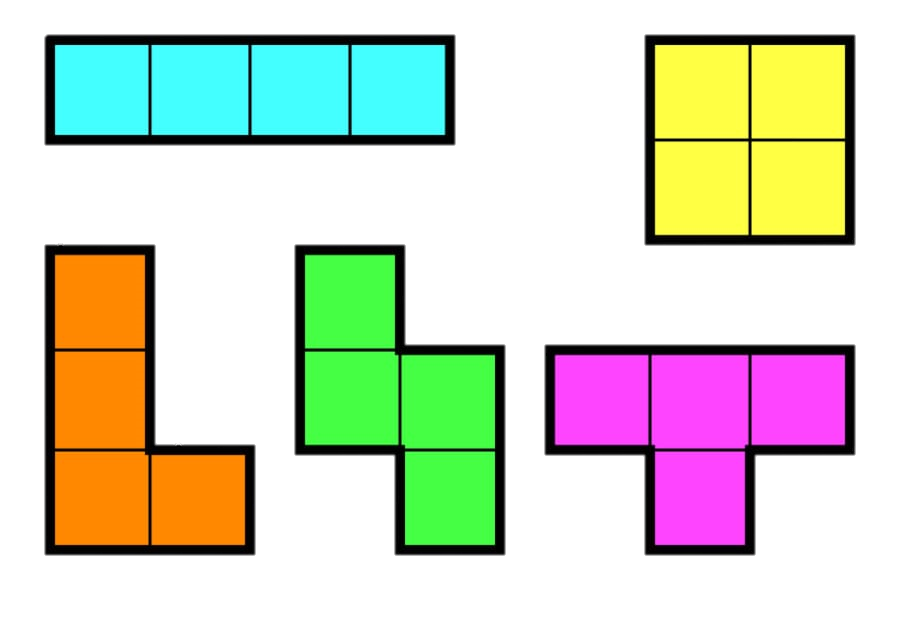
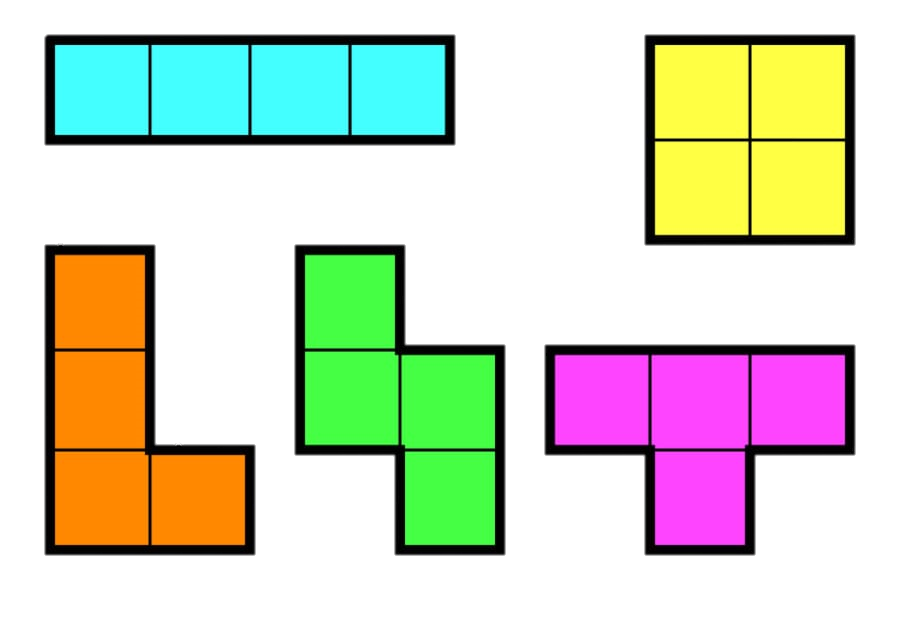
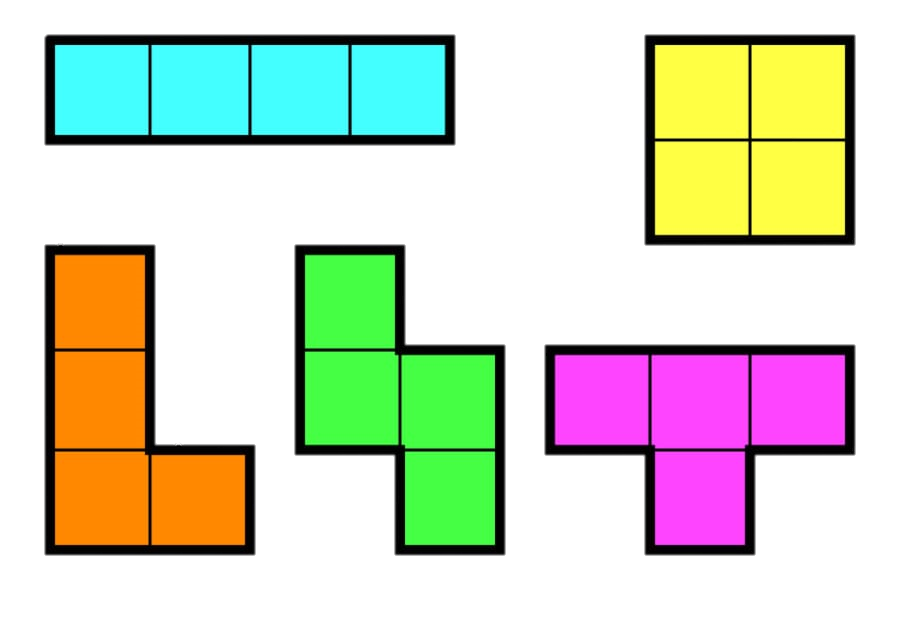
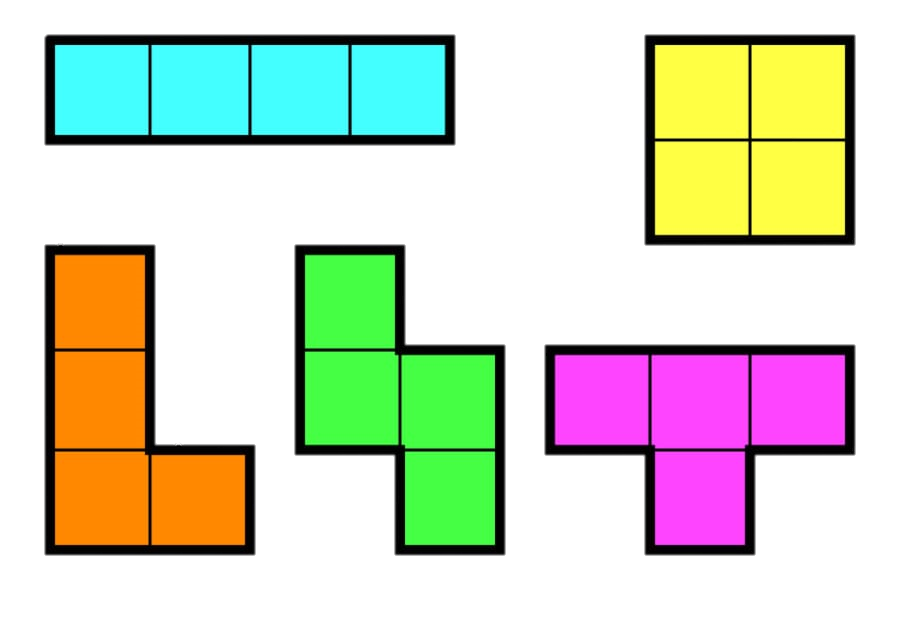
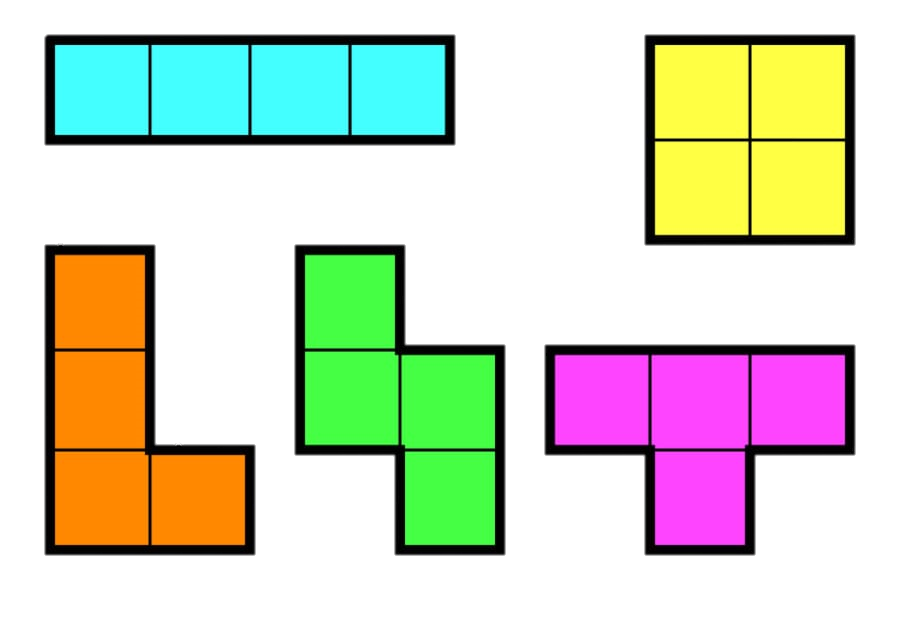
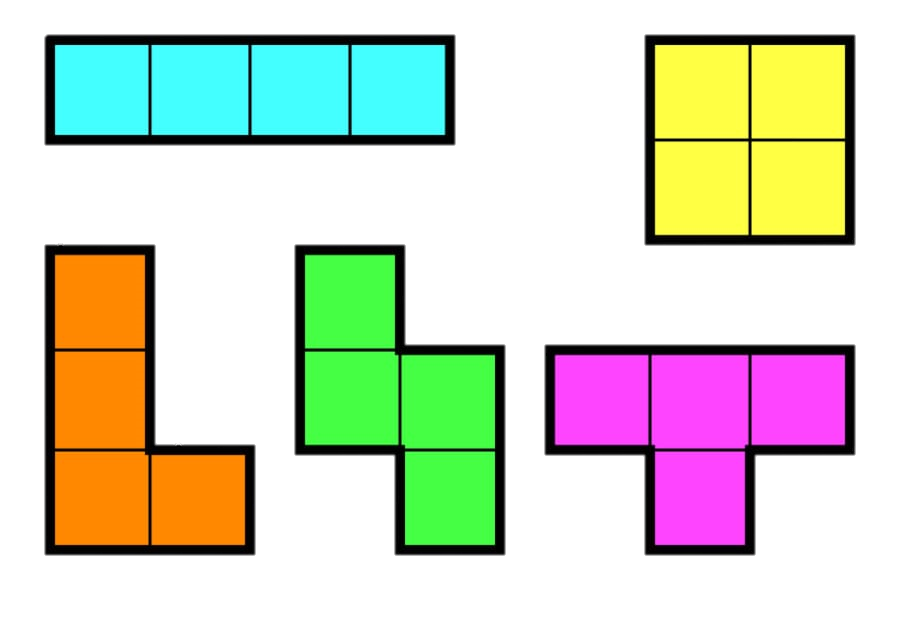
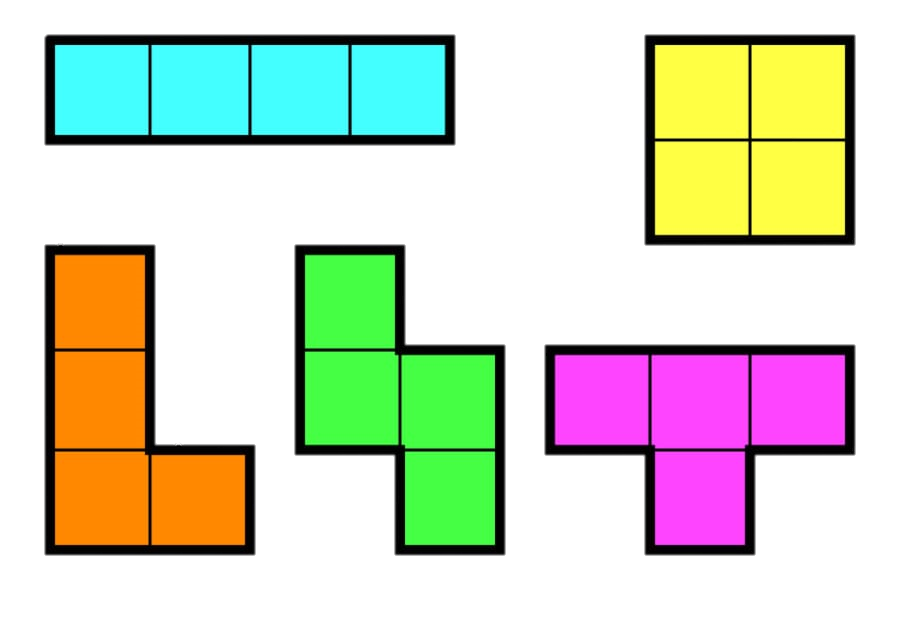
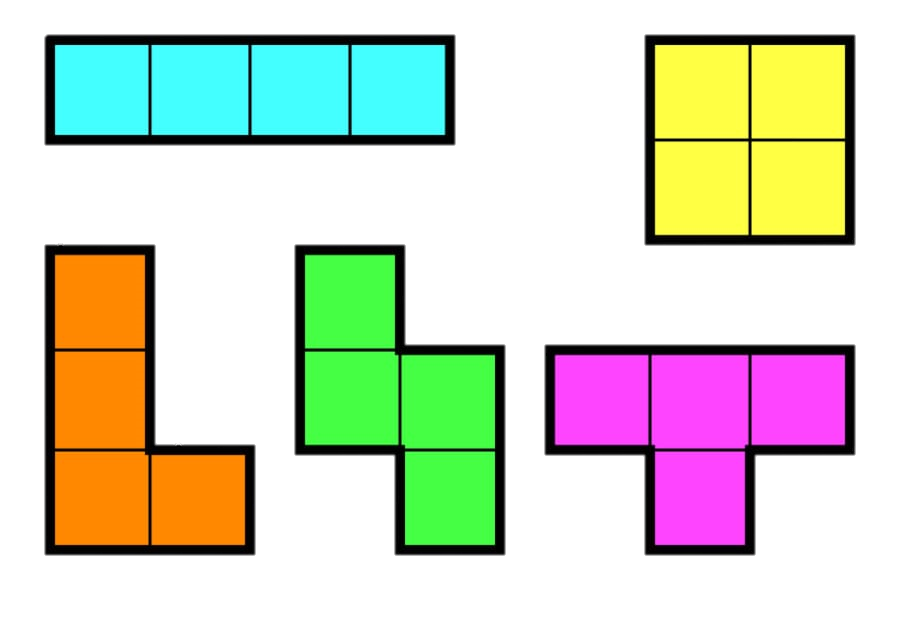
­­­­­­­­­­­­­



**The Tetris project**

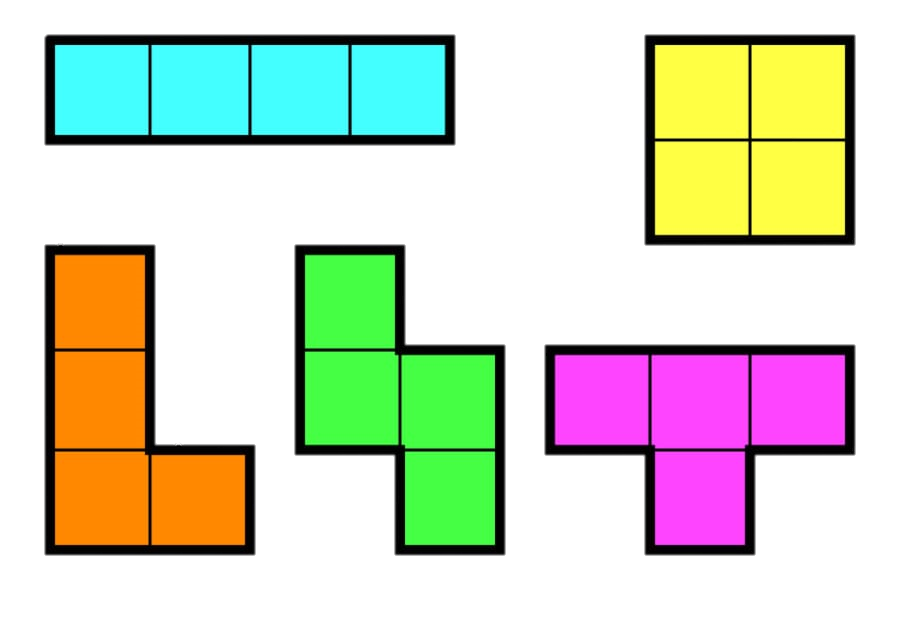
**A Software Requirements Specification document**

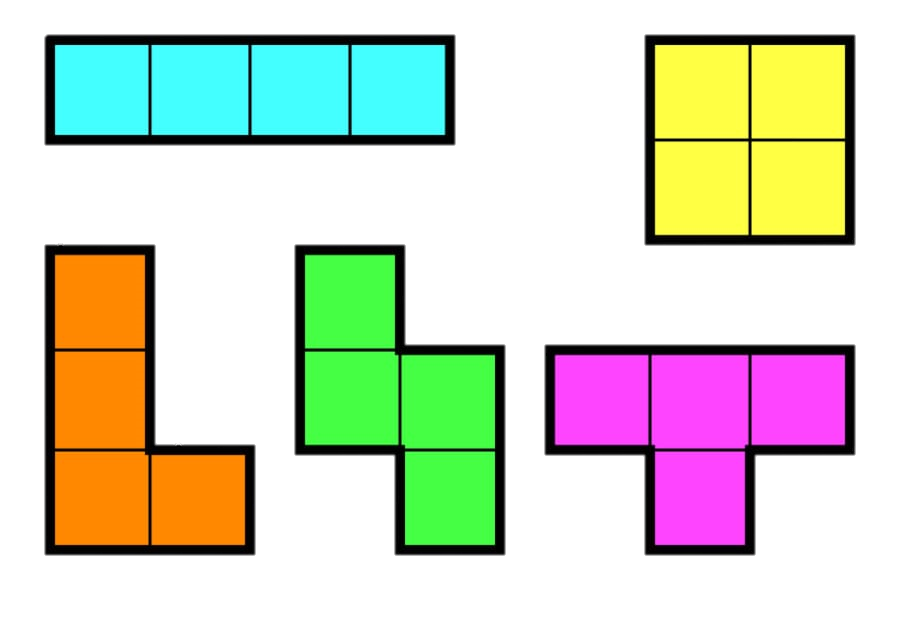
**by:**

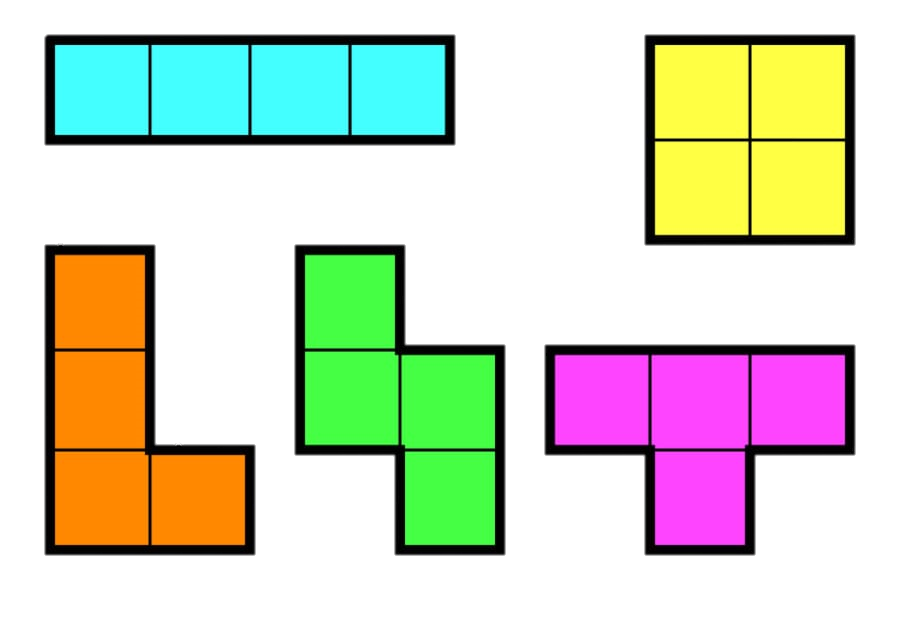
**Ethan Noah, 301741310**

**Dafna Sasson, 201117827**

**Amit Maimon, 311454334**

**Guy Meiri, 201392453**

****

****

**Table Of Contents**

[1. INTRODUCTION 3](#_Toc350265124)

[1.1 Purpose 3](#_Toc350265125)

[1.2 Scope 3](#_Toc350265126)

[1.3 Glossary 3](#_Toc350265127)

[1.4 Figures 1-4 4](#_Toc350265128)

[2. OVERALL DESCRIPTION 6](#_Toc350265129)

[2.1 User Characteristics 6](#_Toc350265133)

[2.2 Product features 6](#_Toc350265133)

[2.3 General Constraints 7](#_Toc350265134)

[2.4 System Components 8](#_Toc350265135)

[2.5 System Architecture 9](#_Toc350265136)

[3. SOFTWARE REQUIREMENTS 10](#_Toc350265137)

[3.1 Software functional requirments 10](#_Toc350265138)

[**3.1.1 Read current Game State** 10](#_Toc350265139)

[**3.1.2 Extract further data** 11](#_Toc350265140)

[**3.1.3 Send key press commands** 15](#_Toc350265141)

[**3.1.4 Simulate moves** 15](#_Toc350265142)

[**3.1.5 Read and write to Database** 16](#_Toc350265143)

[3.2 Required paths of behavior and Message Sequence Charts 17](#_Toc350265167)

[**3.2.1 Update of the game state** 17](#_Toc350265168)

[**3.3.1 Handling of a key press command from the user** 19](#_Toc350265169)

[**3.4.1 Game over scenario** 21](#_Toc350265170)

**3.3 Non-Functional Requirements** ……………………………………………………………………………………………………………… 23

**3.4 External Interface Requirements** …………………………………………………………………………………………………………… 24

**1. INTRODUCTION**

**1.1 Purpose**

The purpose of this document is to specify the software requirements of the "Tetris-AI" project, and to overview the main components and functionality that the software will provide.

**1.2 Scope**

The "Tetris-AI" is a project designed to provide an accessible and reliable API (application programming interface) for programmers who intend to write an AI (Artificial Intelligence) algorithm for the "classic-Tetris" game.

The API will provide functionality for reading game relevant information and sending key press commands to the Tetris game.

**1.3 Glossary**

|  |  |
| --- | --- |
| **Term** | **Definition** |
| API | Application Programming Interface |
| AI | Artificial intelligence – a computer program that simulates a human player of the Tetris game |
| Tetris | The classic Tetris game. |
| the Nintaco Emulator | A Nintendo based game emulator written in Java. The Emulator can run the Tetris game. (Figure 1, page 4). |
| MSC | Message Sequence Chart |
| DB | Database |
| Tetrimino | one of the 7 Tetris Pieces that are used in the Tetris Game (Figure 2, page 4). |
| Game State | An umbrella term that refers to the data of the variables of a game in a specific time point (e.g: score, current Tetrimo, displayed in part in Figure 3, page 5). |
| Tetris Board | The 20X10 matrix on which the Tetris game is being played (Figure 4, page 5). |
| User | The programmer who is writing an AI software for the Tetris game using our platform |

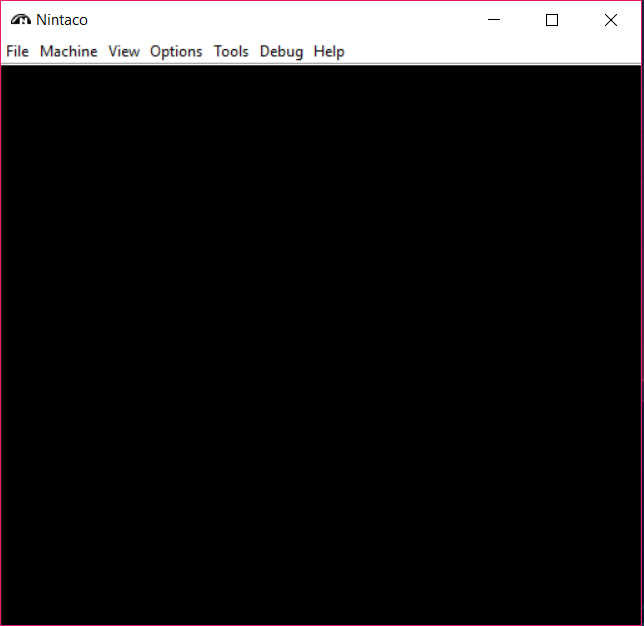


Fig 1. The user interface of Nintaco– the Nintendo Emulator used to run the Tetris game.

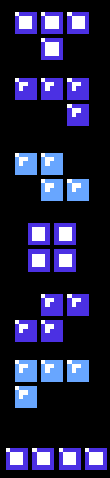
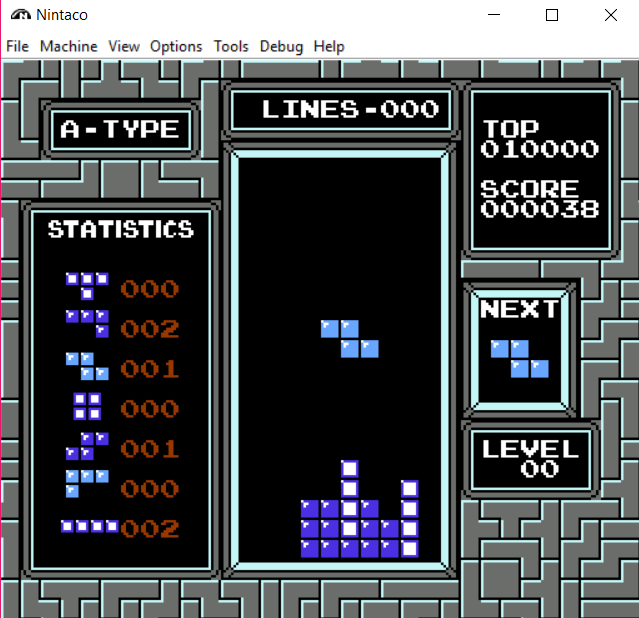


Fig 2. The 7 types of Tetrimino pieces used in the Tetris Game.



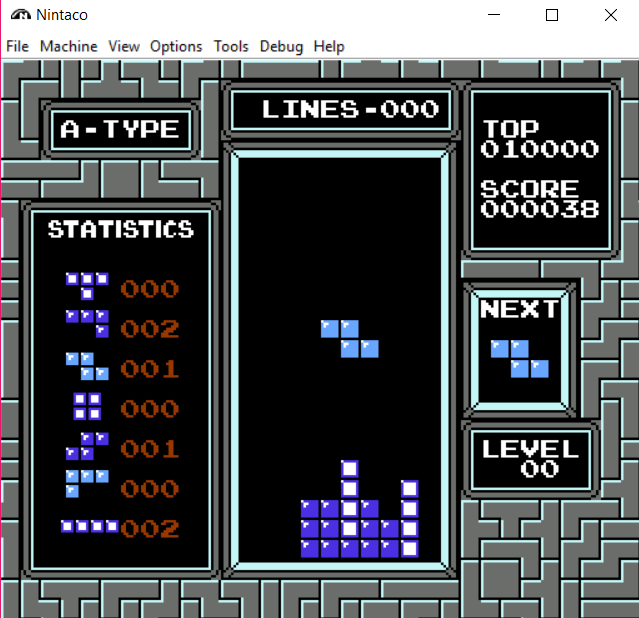
The current score

The next Tetrimo piece to be played

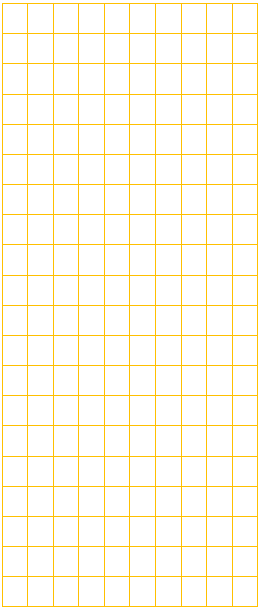
The number of lines cleared so far

The current Level (increased by 1 every 10 lines cleared).

Fig 3. Example of in-game representation of Game state variables described above.



Rows indices

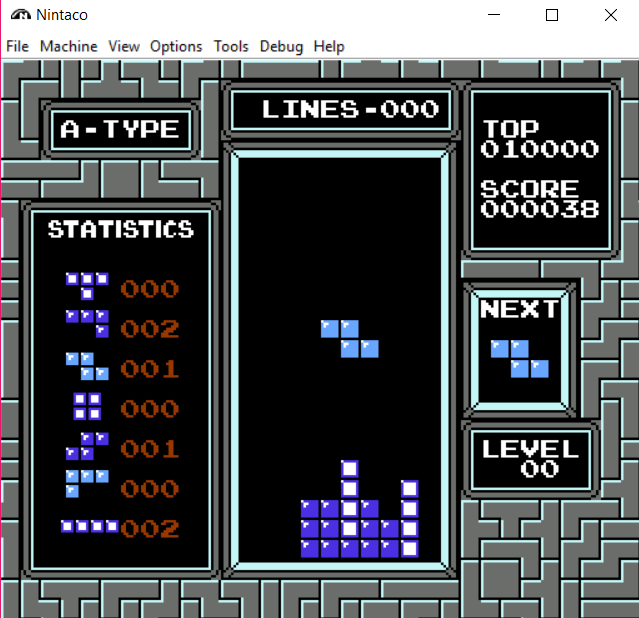


Columns indices

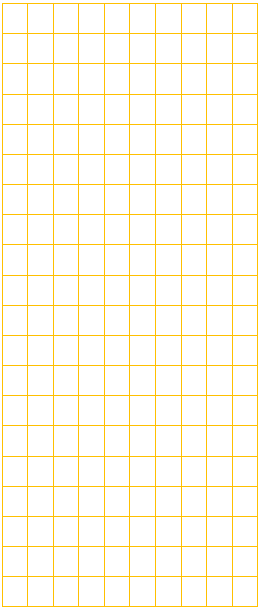


cell at coordinates  
[3,4]

Fig 4. The Tetris board combined with a greed overlay that highlights the 20 rows and 10 columns of the playfield.



Row indices



Column indices



cell at coordinates  
[3,4]

**2. OVERALL DESCRIPTION**

**2.1 User Characteristics**

The end user is any programmer who is familiar with the Python programming language and the Tetris game and is aiming to write an AI (Artificial intelligence) software that will play the game.

**2.2 Product features**

The main component of the project is our Python AI library that enables a programmer to interface with the original Tetris game.  
The product will allow the user to:

* **Receive information regarding the current game state from the game.**

Examples for game state information are the current score and the current tetrimino piece that is being played. (detailed in section 3.1.1).

* **Extract further data from the current game state.**These data are calculated from the current game state and can be used as further input for the AI system.

Examples for these kinds of data are the variance of the column height of the Tetris board and number of “holes” in the Tetris board. . (detailed in section 3.1.2).

* **Send key press commands to the game in real time.**

This feature will allow the AI to make moves in the Tetris game and simulate a human player making decisions. (detailed in section 3.1.3).

* **Simulate moves of the Tetris game without affecting the game being played.**

This will give the AI software programmer the ability to simulate different possible keypresses and their outcome before deciding on which key press command to send to the Tetris Game being played. (detailed in section 3.1.4).

* **Save data of previous games in a Database**

Detailed in section (3.1.5)

* **Query data from previous games from the Database**

Detailed in section (3.1.5)

**2.3 General Constraints**

The original Tetris game was written in 1987 in assembly code which is challenging to interact with and understand. The way we directly interact with the Tetris game is using an API written in JAVA that lets us fetch data from the game memory. A challenge is to figure the memory addresses relevant for the AI software and extract the data correctly for further use.

**2.4 System Components**

**2.4.1 AI**

Artificial intelligence – a computer program that simulates a human player of the Tetris game.   
It receives info regarding the current state of the Tetris game and performs a key press command to affect the game accordingly.

**2.4.2 Tetris**

The classic Tetris game Developed by Alexey Pazhitnov in 1987 for the Nintendo platform.

**2.4.3 Emulator**

A Nintendo based game emulator written in Java. The Emulator can run the Tetris game.

**2.4.4 Python AI library**

The core library that provides game info to the Tetris-AI algorithm and enables it to send key press commands to affect the gameplay.

**2.4.5 Python server**

A network-based mediator between the Java Bridge (defined below) and the Python AI library.

**2.4.6 Java bridge**

The component that is responsible for transferring and collecting the relevant data from and to the emulator which runs the Tetris game.

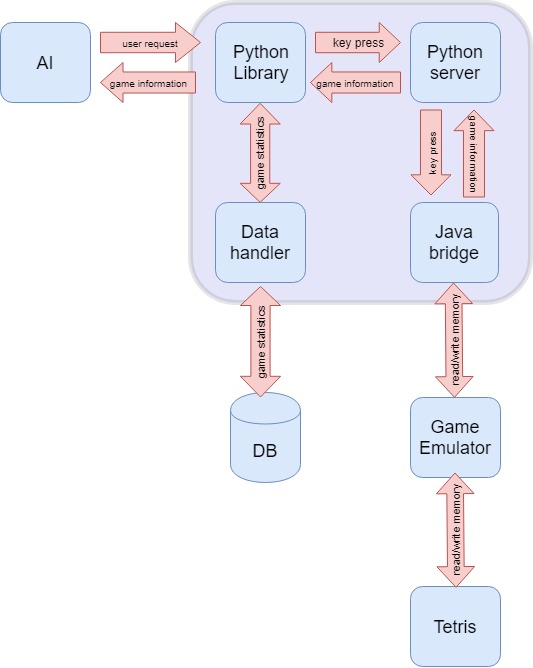
**2.4.7 Data handler**

The component that is responsible for reading and writing from and to the database

**2.4.8 DB - Database**

The Database stores information regarding AI-agents and their performance in previous games.

**2.5 System Architecture**



**3. SOFTWARE REQUIREMENTS**

**3.1 Specific functional Requirements**

Our Python AI library will provide:

* **3.1.1 An ability to read the current Game state .**

This includes data regarding the:

Score

The current score in the game being played.

I.D of the current Tetrimino piece being played

This includes information regarding both the piece type and orientation.

Coordinates of the current Tetrimino piece being played

This includes the X and Y coordinates of the center of the Tetrimino.

I.D of the Next Tetrimino piece

This includes information regarding both the piece type and orientation.

State of the Tetris Board

Information regarding which cells in the Tetris board are occupied and which are empty.

Is Game Over

Information regarding whether the game is still running or not.

Lines cleared

The total number of lines cleared so far by the player.

* **3.1.2 The ability to extract further data from the current game state.**

This includes data regarding:

3.1.2.1 number of filled cells in each row of the board

A “filled” cell is a cell in the board which has a Tetrimino piece locked

to it.

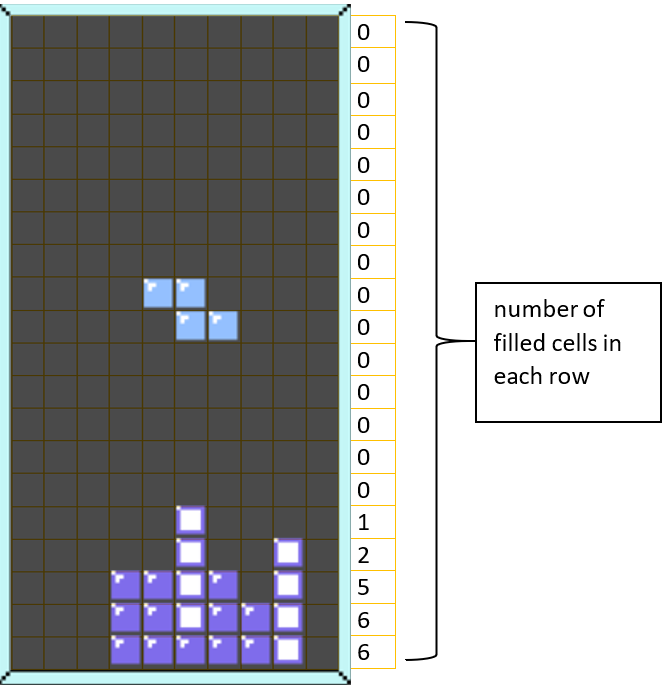


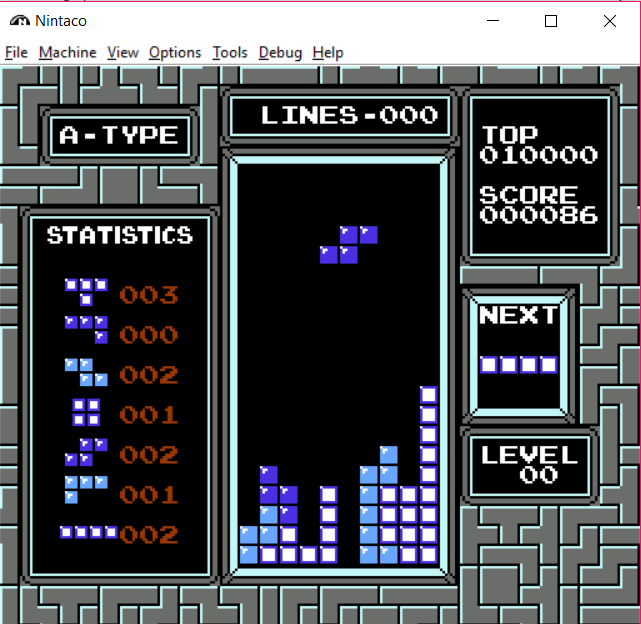
Fig 5. Displayed on the right is the number of occupied cells for each of the rows of the board.

3.1.2.2 total number of wells

A “well” is a column of empty cells which are located above every filled cell in the same column. In addition, the well has only filled cells to its left and right. The left and right most borders of the board (aka the “walls”) are treated as columns of filled cells.

3.1.2.6 total number of “deep” wells

The total number of wells with a length of 3 or more.



“Wells” with depth 3

(AKA: deep wells)

A “Well” with depth 4

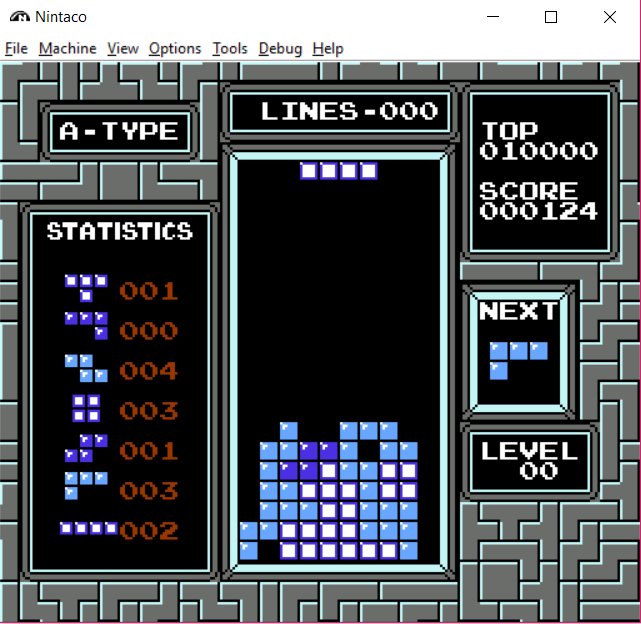
(AKA: deep wells)

A “Well” with depth 2

Fig 6. Example of different types of wells.

3.1.2.3 total number of “holes”

A “hole” is defined as an empty cell which has an occupied cell above it in the same column.



“Holes” in the board

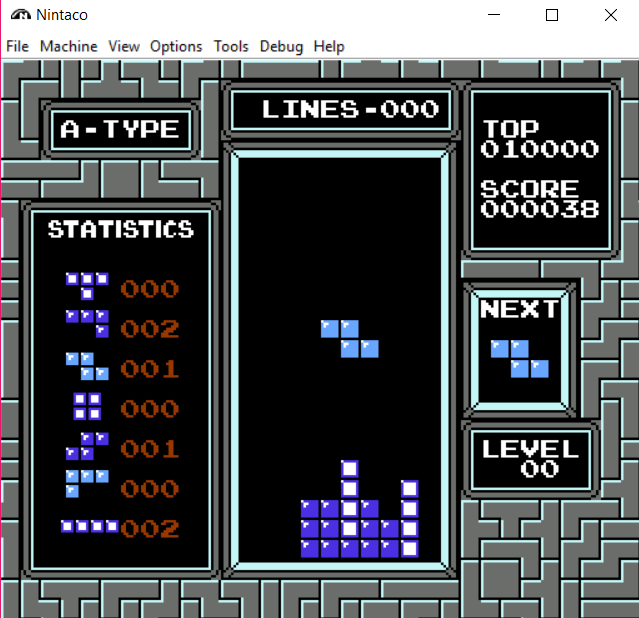
Fig 7. Example of holes in the Board.

3.1.2.4 Column Height variance

The sum of the absolute differences between heights of all adjacent columns.

3.1.2.5 Stack height

The height of the largest column.



Stack Height is 5

(since the highest column is 5 cells above the floor level)

Fig 8. Explanation of column height.

3.1.2.7 Height difference between the highest and lowest filled cells.

This is the maximum stack height minus the minimum stack height.

3.1.2.8 Total Row Transitions

The number of empty cells placed next to a filled cell within the same row. Empty cells adjoining playfield walls are considered transitions. The total is computed across all rows in the board. However, rows that are completely empty do not contribute to the sum.

3.1.2.9 Total number filled cells

The total number of filled cells on the board.

* **3.1.3 An ability to send key press commands to the game in real time**

3.1.3.1 The following keys press commands will be supported:

* + - * Left – if possible, move the Tetrimino one cell to the left.
      * Right - if possible, move the Tetrimino one cell to the right.
      * Rotate clockwise – if possible, rotate the Tetrimino 90 degrees clockwise.
      * Rotate counter clockwise - if possible, rotate the Tetrimino 90 degrees counter clockwise.
      * Down - if possible, move the Tetrimino one cell below it’s current cell.
      * None – send a command that doesn’t do anything. The user could use this command to as a way of letting the current piece “fall” one step in the Y-axis of the board.
* **3.1.4 Simulate moves of the Tetris game without affecting the game being played.**

The user could use the current game state (specified in section 3.3.1), send one of the possible key press commands (specified in section 3.3.3) to be simulated and receive the resulting game state.   
We will provide the user with the ability to visually watch simulated moves using a class that will be implemented in our python AI library.

The AI software could use this feature to evaluate what moves out of the various possible moves lead to a better outcome (in term of the change that they will produce on the game state (specified in 3.3.1) and other parameters derived from the game state (specified in 3.3.2).

* **3.1.5 Saving data of previous games in a Database.**

The system will automatically save information of the game in a database   
once the game is over.

Information that will be stored in the database is the score of previous games of each of the AI software instantiations that used the platform to play with an AI agent.

In addition, the data handler will provide the ability to get further statistics of previous games and agents such as the average score of each one of them.

The database will allow storage of recordings of previous games.  
The user will be able to query the database for previous games and replay them using a class that we will implement (the same class that is mentioned in 3.1.4).  
An example of the graphical visualization of game recordings is featured in Figure 9.

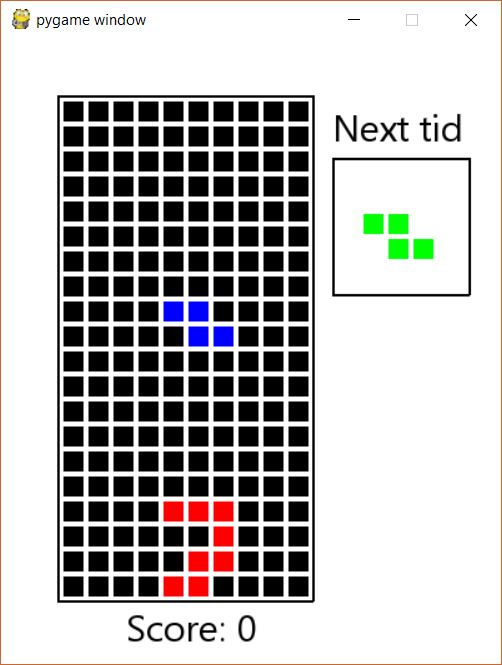


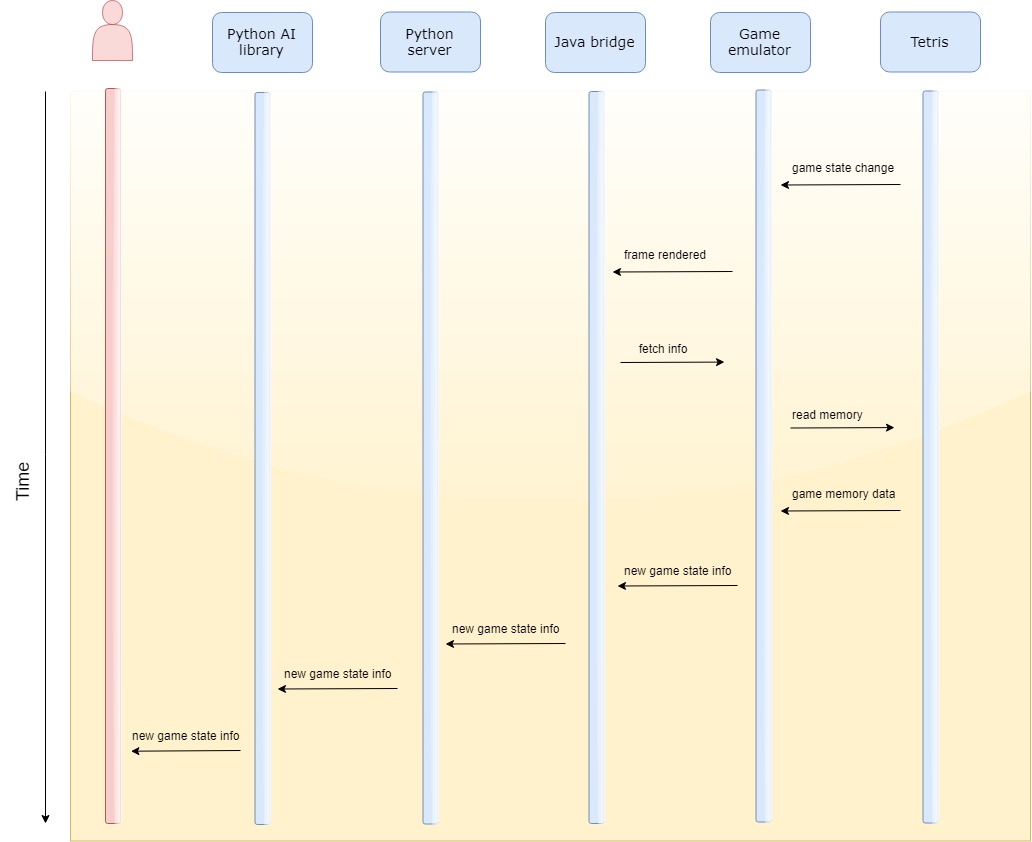
Fig 9. Example of a visualization of a recorded game.

**3.4 Functional Requirements**

**3.4.1 Get an updated Game State**

|  |  |
| --- | --- |
| **Priority** | Essential |
| **Trigger** | A change within the Tetris Game State has occurred |
| **Precondition** | The user has imported the Python AI library |
| **Basic Path** | 1. The Tetris game state is updated in the game memory of the Nintaco Emulator.  2. The Java bridge is informed that a change has been made to the game state and fetches the memory relevant to the new game state from the Emulator.  3.The new Game State is sent from the Java Bridge to the Python server.  4. The server forwards the Game state data to the python AI library.  5. The Python AI library forwards the data to the user (the AI software). |
| **Post condition** | - The user has gained an updated information of the current Game State and uses it to make a decision regarding the next key press to play. |
| **Exceptions** | - Emulator access errors (write / read) |

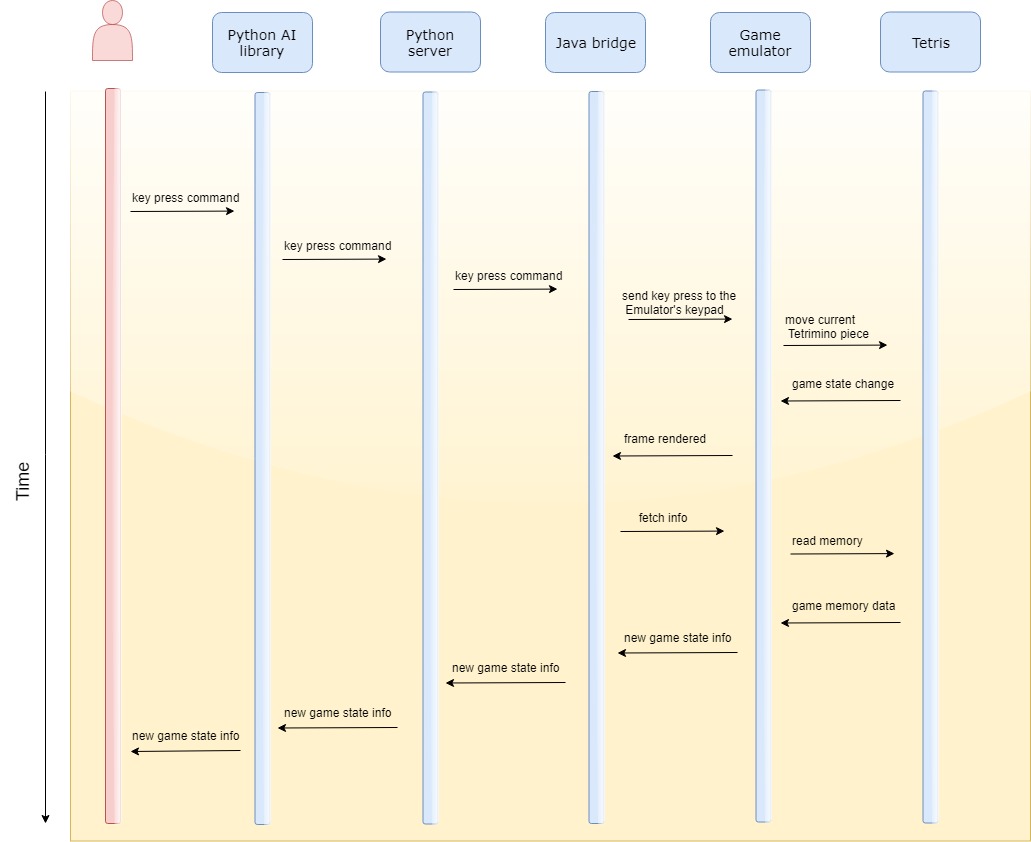
**MSC for an update in the Game State**



**3.4.2 Send a key press command to the Tetris Game**

|  |  |
| --- | --- |
| **Priority** | Essential |
| **Trigger** | The user sends a key press command using a method from the Python AI library. |
| **Precondition** | The user has imported the Python AI library |
| **Basic Path** | 1. The key press command is passed to the Python AI library.  2. The command is forwarded to the Python server which forwards the command on to the Java bridge.  3. The Java Bridge inputs the key press command to the keypad of the Emulator.  4. The Emulator makes a move in the game in accordance to the key press command sent to it.  5. Game state has changed and the updated state is relayed back to the Python AI library and to the user. |
| **Post condition** | - The game has received and played a move based on the decision of the user. |
| **Exceptions** | - Emulator access errors (write)  - Invalid key press command sent |

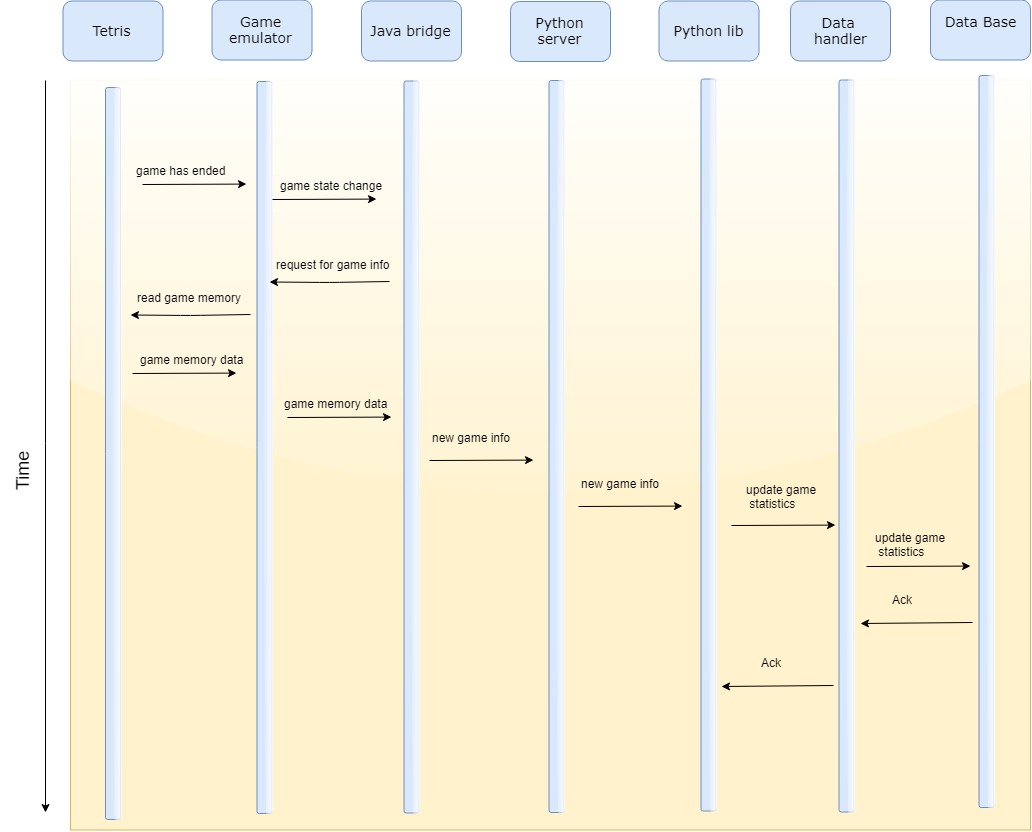
**MSC for a key press command to the Tetris Game**



**3.4.3 Game over - update game statistics in the Database.**

|  |  |
| --- | --- |
| **Priority** | Essential |
| **Trigger** | The game has ended |
| **Precondition** | The user has imported the Python AI library |
| **Basic Path** | 1. The Java bridge is updated that there was a change in the game state.  2. A request for the updated game state is sent from the Java bridge to the Emulator.  3. The game state, including information that the game has ended, is transferred back to the Java bridge and transferred to the Python server.  4. The Python server forwards the data to the Python AI library.  5. The Python AI library sends information regarding the current game state to the data handler.  6. The data handler saves the data in the database.  7. An acknowledgement of the action sent to the Python AI library which forwards it to the user. |
| **Post condition** | - The user has received the requested information |
| **Exceptions** | - DB access errors (write) |

**MSC for Data-base update following a game ending move.**



**3.5 Non-Functional Requirements**

* The Python AI library methods, which are accessible to the user will be documented in regard to their input and output format alongside a general explanation of what each of the methods is intended for.
* Key press commands sent by the user should take effect in the game before the game has progressed to a new game state.
* When an update of the Game state occurs, it should be sent to the user in no more than 0.5 seconds. A delay in the time taken for the user to receive the updated game state will render the information not usable for decision making (since the game will have moved on to another game state by the time the user has a chance to respond to the previous game state).

In order to achieve this goal, we will compress the data ahead of sending it through the network. In addition, we will test the system empirically and check that delay times are no longer than the threshold mentioned above.

* The application shall not change any of the operating system options or make changes to any of the other applications that are installed.
* The application shall not access any data stored on the device except its own DB.

**3.4.4 External Interface Requirements**

**3.4.4.1 User Interface**

Since our end user is a developer using our Python AI library, no graphical user interface is needed.   
The user will use our library in his written code.  
That being said, as described in section 3.1.5 , we will provide the developer with a graphical ability to re-watch past games from recordings.

**3.4.4.2 Hardware Interfaces**

None.

**3.4.4.3 Software Interfaces**

Our Python AI library will use the following software services:

**1. Flask HTTP Server.** Will be used to receive game state update from the Java Bridge component.

**2.** **SQLite3.** Will be used as the database platform of the system.

**3.** **Nintaco.** Described in glossary.

**4.** **The Tetris Game.** Described in glossary.

**5.** **Python 3.** Our Python AI library itself will be implemented in version 3 of the Python language.

**6.** **Java 8.** The Java Bridge component of our system will be implemented in Java 8.