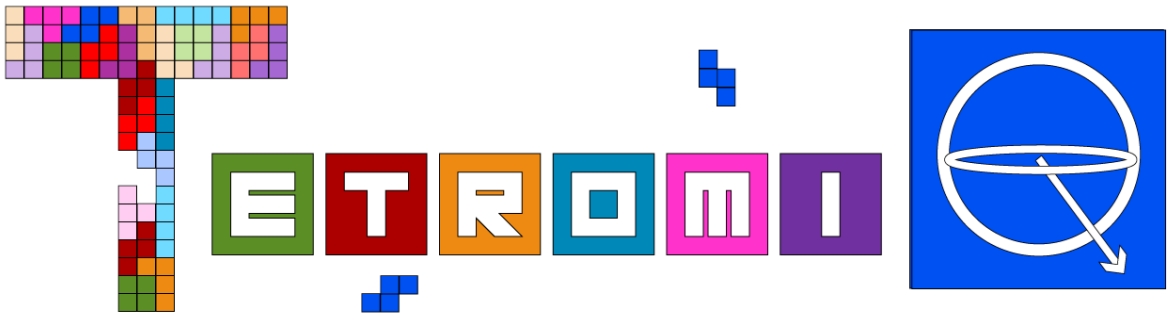


Quantum Games Hackathon 2021

Event organized by the **Quantum AI Foundation** in collaboration with members of **QPoland**.

Contents

Introduction

TetromiQ is a quantum game developed for the Quantum Games Hackathon 2021 that can be played for fun and/or to get familiar with the concept of **superposition**. A *quantum game* is a game that contains a component related to quantum mechanics.

In the case of ***TetromiQ***, a well-known dynamic among video games is used, which involves arranging a series of blocks with different shapes, on a board (playing area) in such a way that no spaces are left uncovered, and points are scored when filling a line from this area.

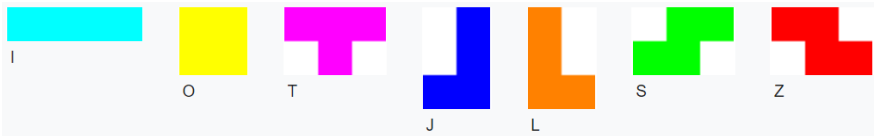
Tetromino

The pieces used in ***TetromiQ*** are known as **Tetrominoes**, they are geometric shapes composed of four squares, connected orthogonally (i.e., at the edges and not the corners), these are a particular type of polynomio, a polyform whose cells are squares. It may be regarded as a finite subset of the regular square tiling [Tetromino (2021)].

A free polyomino is a polyomino considered up to congruence. That is, two free polyominos are the same if there is a combination of translations, rotations, and reflections that turns one into the other. A free tetromino is a free polyomino made from four squares. There are five free tetrominoes.

Free tetrominoes [Tetromino (2021)]

In ***TetromiQ*** we use the one-sided tetrominoes, which are tetrominoes that may be translated and rotated but not reflected. There are seven distinct one-sided tetrominoes. These tetrominoes are usually named by the letter of the alphabet they most closely resemble. The "I", "O", and "T" tetrominoes have reflectional symmetry, so it does not matter whether they are considered as free tetrominoes or one-sided tetrominoes. The remaining four tetrominoes, "J", "L", "S", and "Z", exhibit a phenomenon called chirality. J and L are reflections of each other, and S and Z are reflections of each other. As free tetrominoes, J is equivalent to L, and S is equivalent to Z. But in two dimensions and without reflections, it is not possible to transform J into L or S into Z.



One-sided tetrominoes [Tetromino (2021)]

Tetris

Tetris is a tile-matching video game created by Soviet software engineer Alexey Pajitnov in 1984 for the Electronika 60 computer.

In *Tetris*, players complete lines by moving differently shaped pieces (tetrominoes), which descend onto the playing field. The completed lines disappear and grant the player points, and the player can proceed to fill the vacated spaces. The game ends when the playing field is filled. The longer the player can delay this inevitable outcome, the higher their score will be [Tetris (2021)].

TetromiQ is based on this classic and beloved video game, on the one hand because it is easier to introduce a quantum game based on a classic one, since you can start from an initial understanding of the rules and only add an extension of these so that the player can start playing it in a faster way. And on the other hand, because the creators are fans of *Tetris* and have really enjoyed creating a quantum version of one of their favorite games.

Installation

We recommend the use of a conda environment, to keep the requirements of this game isolated. However, if you don't want to use a conda environment, you can skip to step 3.

1. Create a conda environment:

```
conda create -n tetromiq python
```

Where **`tetromiq`** is only the name of the environment and could be any text with no-spaces.

2. Activate the new conda environment:

```
conda activate tetromiq
```

3. Install what is necessary to run ***TetromiQ***:

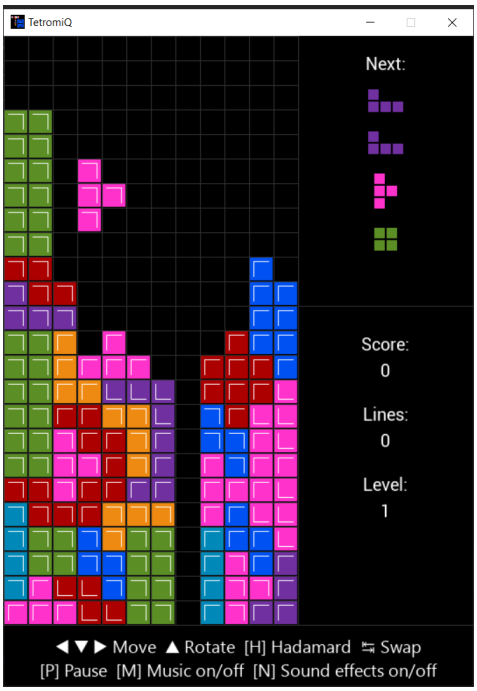
```
pip install .
```

4. Run the game (once in the main game folder):

```
python main.py
```

Game Description

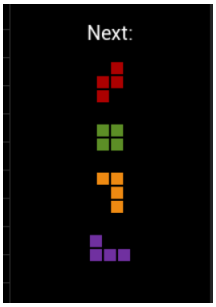
TetromiQ extends the rules of the well-known *Tetris* game, because in addition to creating lines with the one-sided tetrominoes, it is possible to generate quantum blocks, that is, they have an overlap, which will initially cause unexpected behaviors, but over time this probability of having the tile on one side or the other can become part of the player's strategy, so that, as always, they seek to have the highest possible score with each game.



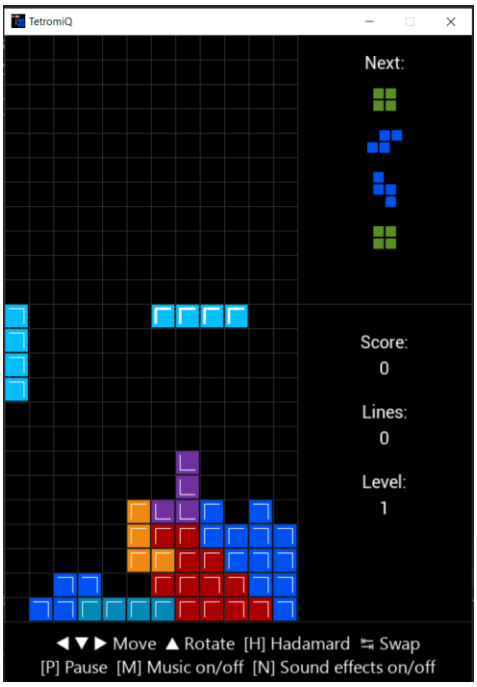
How it works?

Initially, ***TetromiQ*** may seem like a normal *Tetris*, but at first we have a difference, the area of the game is a little larger, both in width and height, this is due to the fact that it is intended that the player has more space (and time) to accommodate the quantum pieces, since when a superposition is generated, two pieces will be going down at the same time, and they should be accommodated.

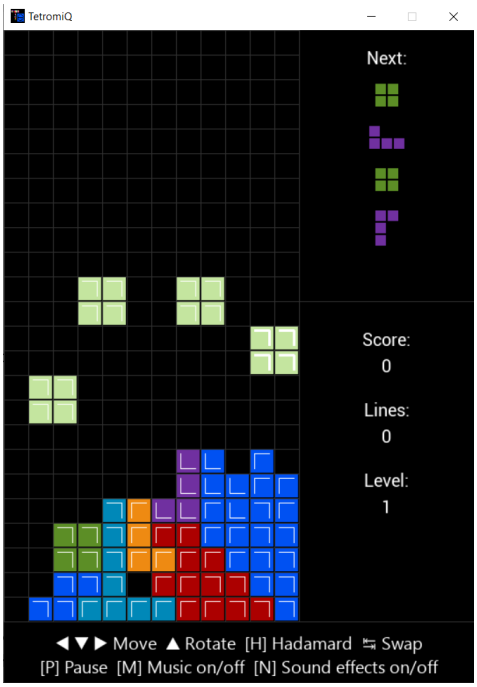
The tiles will come out randomly, and you can see a list of the next ones that will come out, so that you can make better decisions when accommodating the current one (the one that is going down).



TetromiQ mainly uses the concept of quantum mechanics called superposition, when allowing the player to split a block, but this splitting is not normal, it is a division between two parts of the same original block, each sub-block then has half the probability of being the one that remains after a measurement (collapses), so having placed a piece that is in superposition on the board will probably not be there after creating a line (measurement).



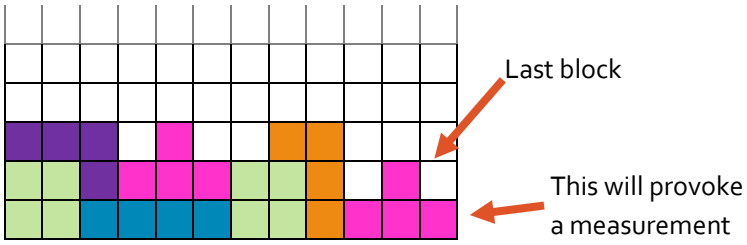
A piece can be divided up to 4 sub-blocks, each with a 25% probability of collapsing when measured (when a line is created):



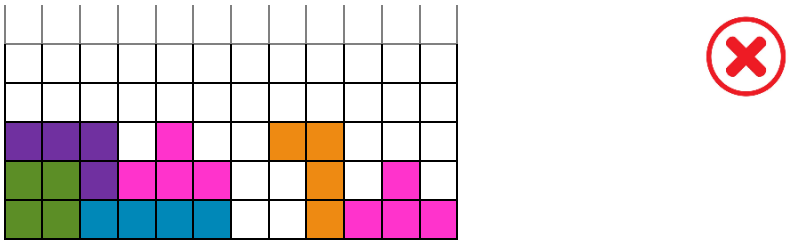
When a piece is split, its color will be clarified to indicate that it is no longer a block that is 100% but that it is 50% or 25% as the case may be.

Working quantum example

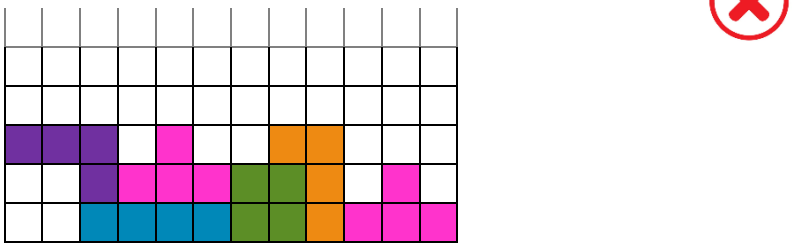
Let's say we have the following scenario, in which a square block was divided in two and its two overlapping parts arranged as shown in the drawing, and then a T-piece reaches the end causing the creation of a line. Because overlapping blocks are involved then a quantum simulation will be run to decide which of the two parts will collapse, then we have two options, whether the left sub-block or the right sub-block collapses, as shown next:



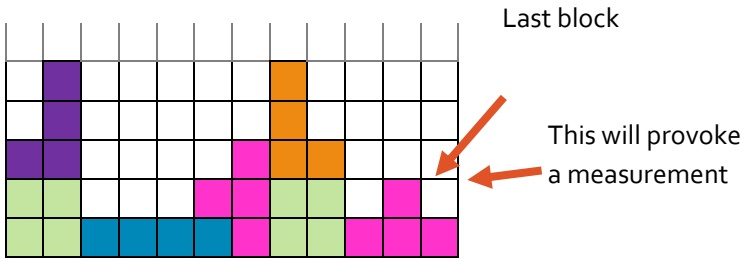
50% chance of left sub-block collapses (no line will be created at the end):



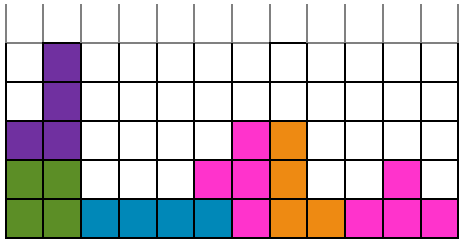
50% chance of right sub-block collapses (no line will be created at the end neither):



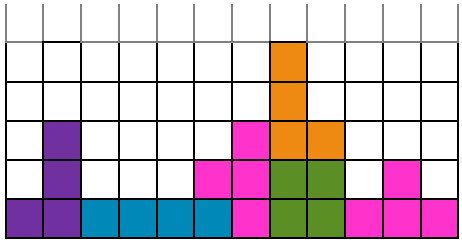
It would have been better to arrange the blocks in this other way which would allow the creation of a line with either of the two collapse options of the square block



50% chance of left sub-block collapses (a line will be created at the end):



50% chance of right sub-block collapses (a line will be created at the end):



Controls and rules

The blocks can be moved with the **arrow keys**, it can be rotated with the **UP** key, it can also be moved sideways with the **RIGHT** and **LEFT** keys, with the **DOWN** key it is possible to accelerate the descent of the tile.

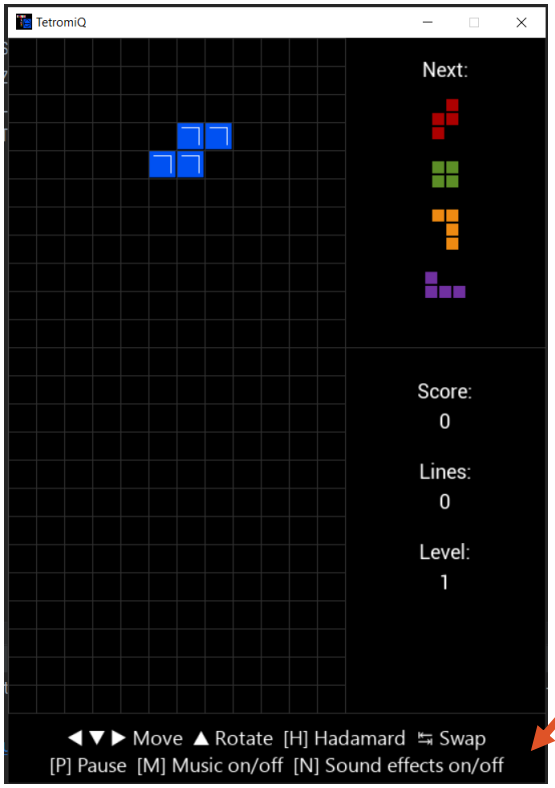
With the **H** key you can split the current piece, put it in superposition, and because now you have two tiles descending at the same time, in order to manipulate one or the other, you must use the **TAB** key to exchange between the two current pieces.

The **H** key has an effect on a normal piece, which is at 100%, and will split it into two pieces that will be at 50% each, and it is possible to use the **H** key a second time, which will place the current tile (the one that is moving at that moment, switch to the other with **TAB**) in superposition again, creating two more blocks with 25% each.

To pause the game momentarily, it can be paused with the **P** key, pressing this key again will allow the game to continue.

This game has music and sound effects, but they can be muted, with the **M** key, the background music is muted, and with the **N** key the sound effects are muted, typing each key a second time will reproduce the sounds again.

The game displays a summary of the use of these keys at the bottom of its window.



Creating a **line** (or several) is the objective of the game, it happens when there are tiles across the board, in such a way that there are no holes in the horizontal, the creation of lines is what increases the score, with the creation of certain number of lines, level is increased.

A **level** is defined by the creation of **10 lines**, so the level is increased each time this number of lines is generated. When you have a level up, the descent speed will increase a little, and after **15 levels**, the speed will decrease again.

Creating a line will increase the **score** by **5 points**, but if the line was generated with the help of an overlapping block that was at 50% at the time and collapses in that position which helps to create the line, then the increase in the score will be **multiplied by 2**, if the involved block was at 25%, then the score to be increased will be **multiplied by 4**. These multipliers are cumulative, so if a line was created with more than one block in its position (that collapses in the position that creates the line), then they will be applied all the corresponding multipliers, so it is very useful to use blocks in superposition, since you can get higher scores.

Quantum Aspects

In the development of this quantum game we covered the concept of **superposition**, and to generate the quantum simulations we have used **Qiskit** and according to the scenarios that happen within the game, one of the four circuits designed for these purposes is executed in a quantum device simulator.

Superposition

Quantum superposition is a fundamental principle of quantum mechanics. It states that, much like waves in classical physics, any two (or more) quantum states can be added together ("superposed") and the result will be another valid quantum state; and conversely, that every quantum state can be represented as a sum of two or more other distinct states. Mathematically, it refers to a property of solutions to the Schrödinger equation; since the Schrödinger equation is linear, any linear combination of solutions will also be a solution.

example is a quantum logical qubit state, as used in quantum information processing, which is a quantum superposition of the "basis states" $|0\rangle$ and $|1\rangle$. Here $|0\rangle$ is the Dirac notation for the quantum state that will always give the result 0 when converted to classical logic by a measurement. Likewise, $|1\rangle$ is the state that will always convert to 1. Contrary to a classical bit that can only be in the state corresponding to 0 or the state corresponding to 1, a qubit may be in a superposition of both states. This means that the probabilities of measuring 0 or 1 for a qubit are in general neither 0.0 nor 1.0, and multiple measurements made on qubits in identical states will not always give the same result [Quantum superposition (2021)].

Qiskit

Qiskit is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules. Qiskit purpose is to accelerate the development of quantum applications by providing the complete set of tools needed for interacting with quantum systems and simulators.

This SDK based on circuits has been the choice to run the simulations within the game since it is easy to use, and being done in Python, the inclusion with the selected library to develop the game has been natural and efficient.

Quantum circuit

In quantum information theory, a quantum circuit is a model for quantum computation in which a computation is a sequence of quantum gates, which are reversible transformations on a quantum mechanical analog of an n-bit register. This analogous structure is referred to as an n-qubit register [Quantum circuit (2021)].

Hadamard gate

In quantum computing and specifically the quantum circuit model of computation, a quantum logic gate (or simply quantum gate) is a basic quantum circuit operating on a small number of qubits. They are the building blocks of quantum circuits, like classical logic gates are for conventional digital circuits. Unlike many classical logic gates, quantum logic gates are reversible.

The Hadamard gate acts on a single qubit. It maps the basis states $|0\rangle \to \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle \to \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ (i.e. **creates a superposition** if given a basis state). It represents a rotation of π about the axis $(\hat{x}+\hat{z})/\sqrt{2}$ at the Bloch sphere. It is represented by the Hadamard matrix [Quantum logic gate (2021)]:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Bell State

In the case of collapsing a piece into two sub-blocks in its position, we use a Bell state that, when measured, will not give one of two options with 50% probability each.

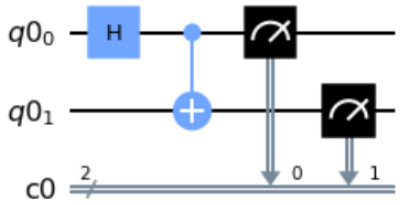
The Bell states or EPR pairs are specific quantum states of two qubits that represent the simplest (and maximal) examples of quantum entanglement; conceptually, they fall under the study of quantum information science. The Bell states are a form of entangled and normalized basis vectors. This normalization implies that the overall probability of the particle being in one of the mentioned states is 1 [Bell state (2021)].

Quantum Circuits for TetromiQ

For each case we use the following circuit to execute the experiment and make the measurement of the blocks in superposition, and thus be able to decide which one is to prevail.

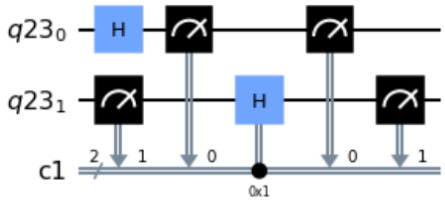
For details of the code that generates these circuits you can consult the **jupyter notebook** that is in the same folder as this file.

CASE 2 BLOCKS: 50%-50%

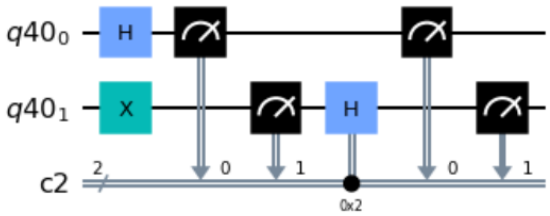



```
{'00': 491, '11': 509}
```

CASE 3 BLOCKS: 50% - 25% - 25%

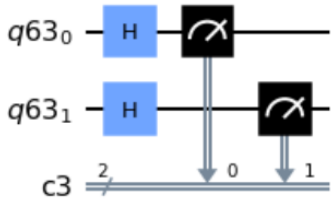



```
{'00': 495, '11': 254, '01': 251}
```




```
{'00': 243, '10': 235, '11': 522}
```

CASE 4 BLOCK: 25% - 25% - 25% -25%




```
{'10': 237, '00': 243, '01': 250, '11': 270}
```

Future Work

During development we have been generating more and more ideas of how this quantum game can be improved and extended, such as forcing the player to use overlapping blocks by sending them from the beginning in overlapping, we also consider that the visual part may have effects that help to understand what is happening, among many other examples, we have added some as issues in the GitHub repository where this project is