TETRIS Documentation

Sandro Chkuaseli

Contents:

USER DOCUMENTATION

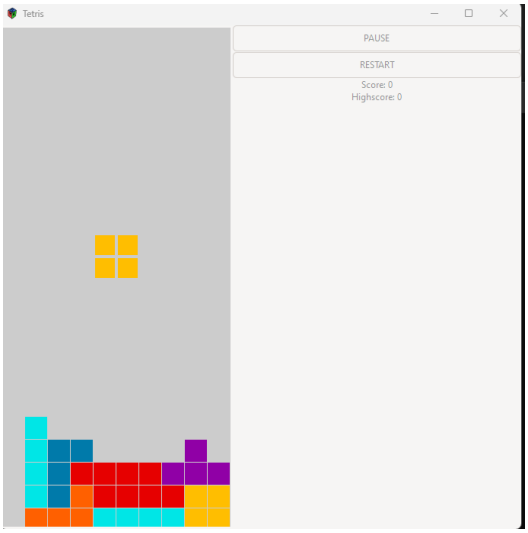
General Description

Tetris is an interactive puzzle game where players manipulate different-shaped tetrominoes as they descend on a 10x20 grid. The player can move and rotate the tetrominoes as they fall on the grid. The objective is to arrange the tetrominoes in a way that completes horizontal lines without any gaps. When a line is completed, it disappears, and the game continues with new pieces falling. The game ends when the stack of tetrominoes reaches the top, meaning the player has lost.

Interacting with the interface

When the player runs the game, they are greeted with an interface made using Gtk#. The interface combines all the necessary elements, including the interactive game grid, control buttons (Pause / Resume and Restart), and labels that track Score and Highest Score respectively. In this Tetris game interface, you'll find:

1. **Game Grid**: On the left side, there's the game grid. It is a playing field where tetrominoes fall, and you can move and rotate them to make lines.
2. **Pause / Resume Button**: It is located on the right-hand side of the window. You can hit this to stop or start the game whenever you want. Pausing freezes everything on the screen.
3. **Restart Button**: It is located below the Pause button. If you want to start a fresh game, just click this button, and it'll clear the board and reset your score.
4. **Score / Highest score Labels**: The score label is located below the restart button and it displays the current score. The score goes up as the player clears lines by filling rows with tetrominoes. The highest score label is located below the score label and it displays the highest score achieved. It updates when the player beats their previous high score.



Controls

Playing the game of Tetris is easy and intuitive as the controls are straightforward. The player must use arrow keys to manipulate the falling tetromino:

1. **Up Arrow:** The up arrow key rotates the falling tetromino clockwise.
2. **Left Arrow:** The left arrow key moves the tetromino to the left.
3. **Right Arrow:** The right arrow key shifts the tetromino to the right.
4. **Down Arrow:** The down arrow speeds up the descent to make the tetromino fall faster.

Other game controls also include the Pause button and the Restart button:

1. **Pause button:** The button stops the tetromino from falling on the game grid until the user presses the button again to resume the game.
2. **Restart Button:** The restart button initiates a fresh game. When the button is pressed the board is cleared and the scores are reset.

As has already been mentioned above the controls are easy and straightforward, making the game user-friendly and easy to interact with.

SOFTWARE DOCUMENTATION

Overview

The Tetris game runs in a single window, it exits either when the window is closed or when the stack of tetrominoes has reached the top meaning the game is over and the message dialog box pops up with a button saying EXIT. The window is divided into two sections.

The left section of the window contains the game grid, which displays the game Tetris. It displays the placed tetrominoes and falling tetromino, the animation on the left is updated frequently as the image being displayed changes a lot when the user is controlling the falling tetromino.

The right section contains pause and restart buttons as well as the score and the highest score labels.

The program is divided into several classes which are written in their respective .cs files.

The interface is created using Gtk#. UI.cs contains all the Gtk# designs and loops.

Classes

GameGrid

GameGrid class is basically a 2-dimensional array where 0's indicate an empty spot and other numbers indicate occupied spots. The GameGrid is initialized by indicating the number of rows and columns you want the grid to have. The GameGrid class includes several methods:

1. **isRowFull():**
    - Checks if a specific row has been completely filled by using the for loop to go over the row.
    - Returns false if it encounters a 0 in the row, otherwise, it returns true.
2. **isRowEmpty():**
    - Determines if the row is empty.
    - Uses same logic as **isRowFull()**
3. **moveRowDown(n):**

- Moves the row down by n rows.
- Copies the numbers from the target row to be moved and sets all 0s in the current row.

4. **clearFullRows():**
    - Scans each row using the for loop and uses isRowFull() method to identify whether it is full.
    - If the row is full it clears it by replacing all values with 0.
    - Keeps track of the number of rows cleared using the local variable int cleared and adjusts the grid using moveRowDown() to maintain a consistent layout.

Tetromino

The Tetromino class contains several crucial properties. A class Position is used to create a 2-dimensional array that will contain the tile locations of all the tiles of tetrominoes in all 4 rotations(there is a class for each tetromino created using Tetromino class properties). The Tetromino class also includes Id property which is later assigned to each tetromino (Id's go from 1-7). We also have a rotationState which indicates in which rotation tetromino is and finally offset which is the offset of the tetromino on the grid. Tetormino itself is initialized using only offset. The class contains several crucial methods:

1. **TilePositions():**
    - The method provides an iterator that yields the exact positions on the game grid that are occupied by the tetromino tiles.
    - It takes into account the current rotation of the tetromino which is a property that I mentioned above.
    - It manages to do this by taking the position of the tile stored in a 2-dimensional array Tiles and adding the offset to get the exact position on the board.
2. **rotateClockWise()** and **rotateCounterClockWise():**
    - Changes the value of rotationState property
3. **Move():**
    - Changes the value of offset to "move" the tetromino
4. **reset():**
    - Resets the state of tetromino by setting rotationState to 0 and offset values to StartPosition(starting offset) value.

There are several classes that inherit from the Tetromino class and define the properties and shape of specific Tetromino (JTetromino class, LTetromino class and so on). All of them are basically the same thus I will be giving a general description of what the class looks like. Each Tetromino is assigned it's own Id, StartPosition and Positions in 4 rotational states.

TetrominoQueue

TetrominoQueue class also inherits from the Tetromino class. This class takes care of providing a new tetromino and queueing them up. The class has a property NextTetromino that hold the next tetromino to be used. It includes one method that neends to be described:

1. getAndUpdate():
    - Retrieves the next tetromino from the queue and updates the queue with a new random tetromino. It doies so by creating a Tetormino which stores the NextTetrtomino, then NextTetromino is updated and the previous value of NextTetromino is returned.

GameState

The `GameState` class is a fundamental component of the Tetris game, responsible for managing the game's state, including the current Tetromino, game grid, score, and high score. This class includes a the following properties:

1. **CurrentTetromino:** Gets or sets the current Tetromino that is actively falling on the game grid. It includes a setter that resets the Tetromino when assigned.
2. **GameGrid:** An instance of the GameGrid class representing the game board/grid where Tetriminos are placed.
3. **TetrominoQueue:** An instance of the TetrominoQueue class.
4. **GameOver:** A boolean property that indicates whether the game is over or still in progress.
5. **Score:** An integer property representing the player's current score.
6. **HighestScore:** An integer property representing the highest score achieved in the game.

GameState initializes a new game state by creating instances of GameGrid and TetrominoQueue. It also initializes CurrentTetromino with the first tetromino from the queue and attempts to retrieve the highest score which is store in a local file.

The GameState class includes a lot of methods as it combines all the other classes we have described earlier to create the game:

1. **rotateTetrominoClockwise():**
    - Rotates the current Tetromino clockwise and checks if the rotation is valid. If not, it attempts to rotate the Tetromino counterclockwise. To do so it uses methods of a Tetromino class.
2. **moveTetrominoLeft()** (we have almost the exact method for moving to the right)**:**
    - Moves the current Tetromino one column to the right and checks if the move is valid. If not, it reverts the Tetromino's position.
    - Uses the Move() method from the Tetromino class.
    - Uses tetrominoFits() method to check whether the move is valid or not.
3. moveTetrominoDown():
    - Exactly the same as move left or right but it also uses the placeTetromino() method to place the block if the move is not valid and it can’t go down any further.
4. **LoadHighestScore():**
    - Loads the highest score from a local text file if it exists. If not, it sets the highest score to 0.
5. **SaveHighestScore():**
    - Saves the highest score to a local text file.
6. **tetrominoFits():**
    - Check if the current Tetromino fits within the game grid without colliding with existing blocks.
    - It checks the tiles using the isEmpty() method from GameGrid class. And uses TilesPositions() method from Tetromino class to get the exact current location of the tetormino.
7. **isGameOver():**
    - Checks if the game is over by examining the top two rows of the game grid for any non-empty cells using isRowEmpty() method from GameGrid class.

8. **placeTetromino():**
    - Places the current Tetromino on the game grid, updates the score and checks for line clearances and game-over conditions.

Position

The Position class represents a two-dimensional position, typically used to describe a location within the Tetris game grid. It contains two properties Row (an int that stores the row component of the position) and Column (an int that stores the column component of the position). Position(int row, int column) initializes a new Position object with row and column values. The class does not contain any methods. It serves as a simple data structure for managing row and column coordinates for the Tetris game in this case.

UI Class

I have already explained and described how the UI works but in this section, we going to go into more detail.

The UI consists of an HBox which holds one vertical and one horizontal box. Horizontal box is where the tetris is being played and the vertical box holds the button and the labels.

The most important part of the UI is probably the Timeouts. To run the game, two timeouts are used. One is responsible for handling the key presses and it ticks every 40ms and the other one is responsible for moving the tetromino down using the MoveTetrominoDown() method of the GameState class, then checking if the user has surpassed the checkpoint to increase the game speed (increasing the game speed is done by shutting down the second Timeout and starting a new one with a decreased tickrate). This timeout also checks if the game is over and if it is displays the message box.

Every button has its separate simple method which I am not going to go into the details of as they are very simple pieces of code.

The most important method is the OnDrawingAreaExposed(object sender, DrawnArgs args) method. The method is responsible for rendering the game grid including the current tetromino on a graphical drawing.

How the method works:

1. Initializes a Cairo drawing context using Gdk.CairoHelper.Create
2. Clears the drawing area using Cairo.Operator.Clear then paint and resets the operator
3. Grid Rendering: It iterates through each cell of the game grid and renders the individual cells based on their id. The id represents the tetromino type or an empty cell (0). For each cell, It calculates the position and size of the cell in pixels based on the row (r) and column (c) indices. Sets the fill color based on the id using a switch statement. Draws a rectangle for the cell and fills it with the specified color.
4. Disposes the Cairo context
5. Rendering the current tetromino retrieved from the gameState.CurrentTetromino
6. Disposes the context again