Programming 2 (SS 2022)
Saarland University
Faculty MI
Compiler Design Lab

Prof. Dr. Sebastian Hack
Julian Rosemann, B. Sc.
Marcel Ullrich, B. Sc.

Project 5

Tetris (18 Points + 2 Bonus points)

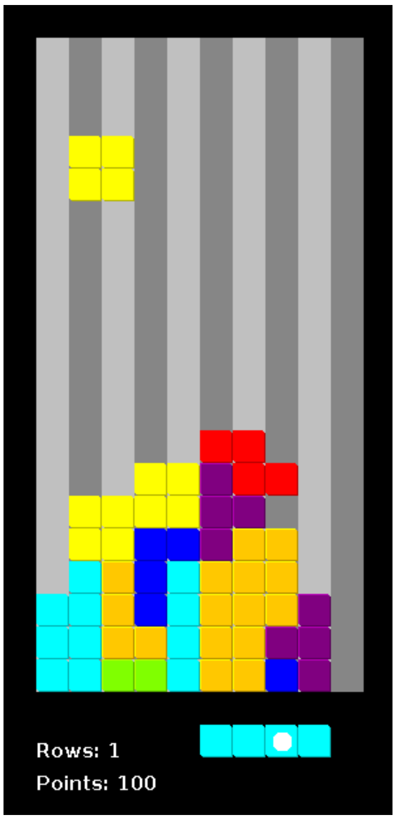
In this project you have to program the game Tetris. Note, you have to pass all public tests for each exercise to get any points for the respective exercise.

1 The game

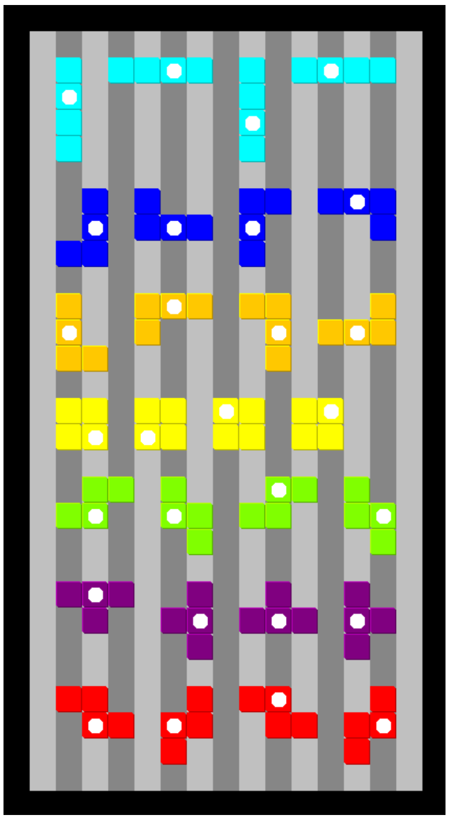
The game is played on a grid-like `Board`. During the game, varying game `Pieces` fall from the top. The goal is to fill the rows with as few gaps as possible by rotating and horizontally moving the piece. To get to know the game's principles better, you can look up further description of the game or test one of the freely available implementations on the internet.[1]

2 Assignment

Your Assignment is to implement the game logic. This includes implementing the `Pieces`, the game's `Board` as well as the game itself (`TetrisGame`). Additionally, an automated player shall be implemented that is able to play Tetris on its own.



(a) Tetris game view (`TetrisComponent`)



(b) Tetris pieces view (`PieceComponent`)

Figure 1: Provided views on Tetris.

[1] `https://en.wikipedia.org/wiki/Tetris`

3 Pieces (3 Points)

Each `Piece` comprises four connected fields. There are seven distinct shapes (`I, J, L, O, S, T, z`). The name of a piece is based on the shape of the letter. Figure 1b shows the possible shapes. The pieces are created by the `PieceFactory`. Each piece has the following properties and concepts:

Body Every piece is defined by a two-dimensional *boolean* Array, whereas the first dimension represent the rows and the second, the columns. The point $(0, 0)$ is on the top left. Each position has the value *true*, if this position is filled by the piece. The array must always be only as high and wide as necessary to accommodate the piece.

Point of rotation Every piece has a point of rotation. It defines the position that the piece can be rotated around. When a piece is placed on the board, it always receives the position, at which the point of rotation shall be placed. In figure 1b the points of rotation are visualized.

Rotation Every piece can be rotated around its point of rotation. This can be done either clockwise (`getClockwiseRotation()`) or counter-clockwise (`getCounterClockwiseRotation()`).

After four rotations, every piece is back in its initial orientation. If the piece type (initial shape), the orientation and the point of rotation are equal, the `equals(Object obj)` method (see `java.lang.Object`) shall return *true*.

Type Every piece has a type (`PieceType`) that stands for the base shape from which the piece has originated through rotation.

Copyability Every piece shall provide the method `clone`, which returns a deep copy of the piece. Note, this means a deep copy of the two-dimensional array that represents the piece, has to be made. Therefore, every value has to be copied, not just the reference to the array.

4 Playing board (4 Points)

The playing board is a two-dimensional grid, on which the pieces can be placed and removed. As with the pieces, first the row and then the column is addressed and $(0, 0)$ refers to the top left corner.
Figure 1b shows the view generated by `PieceComponent`. If the following parts are implemented, you can use the figure to compare against your own implementation. Run the `main` function from `PieceComponent` as Java application.[2] The prerequisites are:

- the classes `Piece` and `PieceFactory`,
- the methods `getBoard()` and `getNumberOfRows/-Columns()` of the `Board`,
- `createPieceFactory()` and `createBoard()` in `MyTetrisFactory`.

The following specifies the respective behavior:

Place piece A piece is placed on the playing field by specifying a shape and a position (row, column). The position thereby means the position of the point of rotation. The method `addPiece()` places a given piece on the board or throws a `IllegalArgumentException`, if that is not possible. The method `canAddPiece()` can be used to check whether a piece can be placed on the board at a given position.

Remove piece Removing pieces is analog to placing pieces on the board. `removePiece()` removes a piece. Again a corresponding `canRemovePiece()` method can be used to check whether a piece can be removed.

Complete rows The method `deleteCompleteRows()` checks whether any row on the board is filled. If that is the case, these rows are removed from the board and all rows above are completely moved down by one.

Copyability Your implementation of `Board` should provide a method `clone`, that performs a deep copy of the playing board, as already required for `Piece`.

[2] In Visual Studio Code, you can do so by right-clicking on the respective file and select `Run Java`, selecting `Run` on top of the main function or clicking the arrow at the top of the editor.

5 The game (5 Points)

The interface `TetrisGame` represents the tetris game. A class implementing this interface shall manage at least the following fields:

Board The playing board that is played on. Initially the playing field is empty and only after a call to `newPiece()` or `step()`, does `currentPiece` have a value. If `step()` is called while the playing board is empty, it shall behave exactly as `newPiece()`. If placing the first piece is not possible, the behavior is undefined.

Pieces and position The `currentPiece`, its position (`pieceRow`, `pieceColumn`) as well as the `nextPiece`. The next piece shall be set initially.

Rows and points The points already achieved in the game and how many rows were completed. Use the following pattern for the points: if in any step, one row is completed 100 points are given, for two rows 300, for three 500 and for four 1000 points.

Additionally, the interface provides the following functions:

Moving a piece The current piece can be moved a field to the right, left or bottom, if that is possible.

Rotation of a piece The current piece can be rotated. This is possible only if the target position of the rotation is not occupied already. Specially, obstacles on the path do *not* have to be taken into consideration.

New piece The method `newPiece()` adds a new piece (`nextPiece`) to the board and randomly selects a next piece. The position of the new piece is

(2, `board.getNumberOfColumns()`/2).

Before the new piece is added to the playing field, it is checked whether any rows on the board are completed. If this is the case, the corresponding methods are called. If the new piece cannot be placed on the board, the game is over (game over).

Game step The function `step()` moves the current piece one row down. If that is not possible, a new piece is added to the game with the `newPiece()` method.

You can start the class `BoardComponent` or use the method `createBoardView()` in the class `Views` to test your implementation of the playing board. The first variant shows an empty playing field on startup, the other shows the given playing field (see figure 1a).

5.1 Observer

The interface `TetrisGame` extends the interface `GameObservable`. Your implementation therefore has to inform all registered `GameObservers` if any of the following events happen:

Change of the position When the position or rotation of the current piece is successfully changed, the observers' method `piecePositionChanged()` has to be called. This happens for every update, i.e. if the piece is moved twice to the left, the method is called twice as well.

Landing of a piece When the current piece landed, i.e. it can not be moved further down, the observers' method `pieceLanded()` is called. This happens in the `step()` method before it is checked whether there are completed rows and before the new piece is added to the board. The current piece is not updated at this point in time, yet.

Completed row When the landing of a piece completes a row, the observers' `rowsCompleted()` method is called.

Game over When the game is over, i.e. no new piece can be added to the board in the `step()` method, the observers' `gameOver()` method is called.

6 Autoplayer (6 Points)

An autoplayer can play Tetris automated. To do so, it implements the `GameObserver` interface to get informed about changes in the game. Additionally, it implements the `AutoPlayer` interface. As long as the game is not over, the method `getMove()` is used to query the move of the autoplayer. The possible moves are listed in the `AutoPlayer.Move` enum. In this mode, the autoplayer is in control of when the next `step` is performed, by selecting `Move.DOWN` as next move. The class `AutoplayerView` is used to start the game. `getMove()` is repeatedly called, and the returned move is executed until the game is over. In case a move is not possible, the game is terminated. The autoplayer must exclusively use the given interfaces and must *not* modify the game's state directly.

6.1 Bonus: Beat the tutors (2 Bonus points)

You can get two additional points, if you manage to beat the autoplayer of the tutors. For the duel, we will let your autoplayer compete against the one of the tutors using the same random seeds.

6.2 Autoplayer tournament

At the end of the lecture, a tournament will take place, where your autoplayer competes against those of your fellow students. There will be an additional document stating the tournament's rules. Note, participating at the tournament is on a voluntary basis and does not give you additional points.

7 Playing Tetris

You can play your own Tetris game by executing the file `PlayerView.java` as a Java application. Using the arrow keys *left*, *right* and *down*, you can move the pieces and using *q* and *w* you can rotate those (counter-) clockwise respectively. You need to implement everything except the autoplayer for the game to work properly.

8 Custom tests

You can add additional tests in the package `tetris.tests`, which will tested against the reference implementation on our server. Your own tests are not graded.

Project setup

To be able to edit the project in Visual Studio Code, you first have to checkout the repository and import the project:

1. Clone the project in any folder:
   `git clone https://prog2scm.cdl.uni-saarland.de/git/project5/$NAME /home/prog2/project5`
   where $NAME has to be replaced with your CMS username.
2. Open the cloned directory in Visual Studio Code and confirm that the contents are trusted.

Remarks

- We use factory methods from the class `MyTetrisFactory` to test your implementation.
- Do not change existing files in the project as they will be reset to the original state for the tests. This does not apply to `MyTetrisTest.java` and `MyTetrisFactory.java`, which you may use for your tests and your implementation respectively. You may add additional classes in the packages that start with `tetris.`

Good Luck!