Com S 228
Fall 2010
Programming Assignment 1
Due at 11:59 pm Friday, September 17

This assignment is to be done on your own. If you need help, see me or one of the TAs. Please make sure you understand the "Academic dishonesty" section of the syllabus.

Please start the assignment as soon as possible and get your questions answered *early*. Read through this specification completely before you start. This is a new project so there are likely to be problems. *Check regularly for updates and clarifications.*

WebCT

The WebCT discussion for Assignment 1 is a good place to post general questions. Please do not post or attach any source code for the assignment. Also check the "official clarification" thread for corrections or updates to this document.

Introduction

The purpose of this assignment is to give you a chance to review some of the basic skills you learned in Com S 227 and to give you some practice working with interfaces and inheritance.

In this project you will complete the implementation of a simplified version of a Tetris-style or "falling blocks" type of video game. This particular game, which we'll call CS228BlockGame, was invented for this assignment and is more or less a cross between Tetris and Columns. If you are not familiar with such games, you can read about them on Wikipedia:

http://en.wikipedia.org/wiki/Tetris
http://en.wikipedia.org/wiki/Columns_%28video_game%29

The basic idea is as follows. The game is played on a grid with 24 rows and 12 columns. Each location in this grid has a position that can be represented as an (x, y) pair (its column and row). We typically represent these positions using the simple class java.awt.Point. At any stage in the game a grid position may be empty or may be occupied by an icon, which for this game is just a colored square, or *block*. In addition, a shape made up of a combination of blocks, called a *polyomino*, falls from the top of the grid. This is referred to as the *current* polyomino. In general the current polyomino can be **shifted** from side to side using the arrow keys, it can be **transformed** using the up-arrow key, which flips it across its vertical axis, and hitting the space bar will **cycle** the blocks within the shape (that is, change the positions of the colors). In addition, the down-arrow can be used to increase the falling speed.

When the currently falling polyomino can't fall any further, its blocks are added to the grid, and the game checks whether it has completed a *collapsible group*. For this game, a collapsible group is any set of three or more adjacent blocks of the same color. (Diagonal rows are not considered collapsible groups.) All blocks in collapsible groups are removed from the grid and blocks above them are allowed to fall (as if by gravity). The new block positions may form new collapsible groups, so the

process is repeated until there are no more collapsible groups.

You might want to note that this game differs from the standard version of Tetris in the following ways:

- An occupied position in the grid may have empty cells beneath it.
- The game may start out with some of the cells occupied.
- In Tetris, the polyominoes can be rotated 90 degrees at a time; in this game, they can only be flipped side-to-side
- In Tetris, all blocks in a polyomino are the same color, so the cycle operation has no effect
- In Tetris, a collapsible group consists of any completely filled row, regardless of the colors of the blocks.

Summary of tasks

The tasks to be completed by you are briefly summarized below.

1. Implement the three concrete polyomino classes described below that implement the IPolyomino interface, including an abstract class Polyomino containing common code for the three concrete types
2. Implement a reasonably comprehensive set of Junit4 tests for the LTetromino only. Put your test code in a package called edu.iastate.cs228.hw1.test
3. Implement a class BasicGenerator that implements IPolyominoGenerator
4. Implement the transform(), shiftLeft() and shiftRight() methods in the AbstractBlockGame class.
5. Create a concrete subclass CS228BlockGame extending AbstractBlockGame that implements the game described in this document, that is, implement the methods determineCellsToCollapse() and determineScore().
6. Modify the create() method of GameMain to create a CS228BlockGame instead of a SampleGame.

The user interface for the project, consisting of the classes in the edu.iastate.cs228.hw1.ui package, uses the Java Swing libraries. However, all the Swing code is already implemented so it is not strictly necessary for you to read and understand it. (However, you might find it interesting!)

Except as noted in the task list above, *no existing code in the project should be modified* for this assignment. You should put all your new classes in the package edu.iastate.cs228.hw1.impl.

The IPolyomino interface

See the javadoc for the IPolyomino interface.

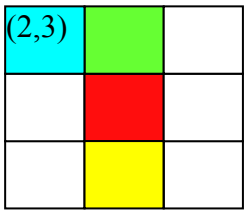
The currently falling shape is represented by an object that implements the IPolyomino interface. Each polyomino has a state including

- The position in the grid of its bounding square
- The actual icons or blocks that make up the shape
- The actual locations in the grid of the blocks (which can change depending on the transform()

operation)

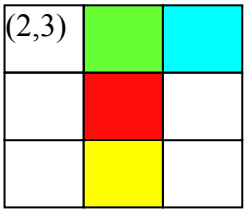
The position of a polyomino is always the upper left corner of its bounding square. Most importantly, there is a getCells() method that enables the caller to obtain the *actual* locations within the grid of the blocks in the polyomino.

For example, suppose we have the tetromino (polyomino with four blocks) shown below in its initial (non-flipped) configuration at position (2, 3). (Note that the colors are normally assigned randomly by the generator for the game.) Then the getCells method should return four cells with locations (2,3), (3,3), and (3, 4), and (3, 5).

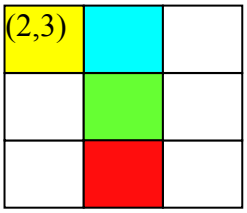


The individual blocks in the grid are represented by instances of IGameIcon. In this case the "icons" are just colored blocks (implemented by the Block class). A block just consists of a *color* and a *marked* status (used by the game to indicate that the block is about to be collapsed). Note that getCells needs to return information about the actual blocks, as well as their locations. Therefore getCells returns an array of the type Cell, which simply encapsulates a block and a location.

Note that if the polyomino in the figure above is transformed (flipped), the location of the bounding square does not change, but the returned cells will be different, as shown:



Likewise, if the cycle() method is invoked, the locations for the cells stays the same but the blocks associated with the locations will change. The illustration below shows the result of invoking cycle() on the first figure:

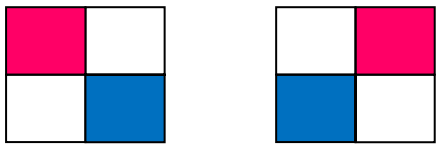


There are three polyominoes to implement:

1. The one illustrated above, called the LTetromino.

2. The ITriomino, which has a 3 x 3 bounding square with the blocks down the center at (1, 0), (1, 1), and (1, 2). Note that the initial and flipped configurations are identical. There is an illustration of a falling ITriomino in the screenshot on the next page.

3. The SlashDomino, which has a 2 x 2 bounding square, shown below with its initial configuration on the left and its flipped configuration on the right.



When you implement these three concrete types, pay careful attention to code reuse, and implement common code in an abstract superclass called Polyomino.

The IGame interface and AbstractBlockGame class

See the javadoc for the IGame interface.

The class AbstractBlockGame is a partial implementation of the IGame interface. The GUI interacts with the game logic only through the interface IGame and does not depend directly on the AbstractBlockGame class or its subclasses. The key method of IGame is step(), which is called periodically by the GUI to transition the state of the game. The step() method is fully implemented, and it is not necessary for you to read it unless you are interested. However, there are three methods within the AbstractBlockGame class that you'll have to implement. Look for the TODO markers in the source code and read the IGame javadoc carefully. You should not modify anything else in IGame or AbstractBlockGame.

AbstractBlockGame is a general framework for any number of Tetris-style games. It is specialized by implementing two abstract methods.

`List<Point> determineCellsToCollapse()`
: Examines the grid and returns a list of locations to be collapsed.

`int determineScore()`
: Returns the current score.

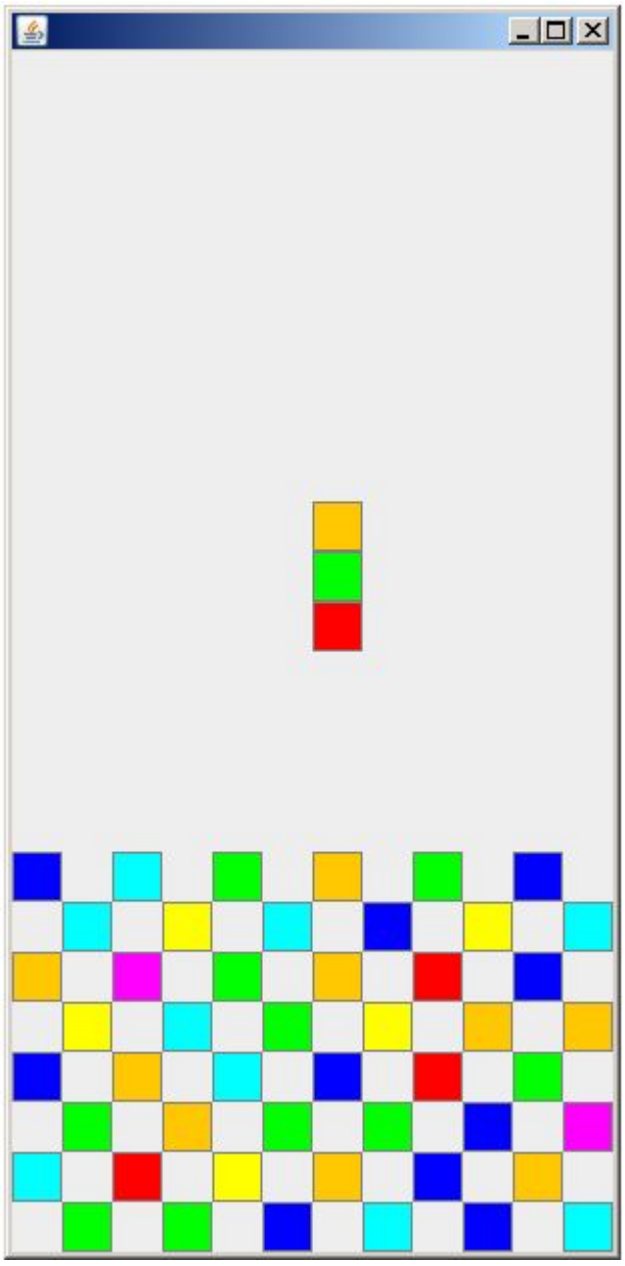
As an example, the SampleGame class is an extension of AbstractBlockGame implementing a simple form of Tetris. That is, determineCellsToCollapse() returns all cells that form a completed row, and determineScore() returns the total number of rows completed so far in the game.

The CS228BlockGame class

Create a subclass of AbstractBlockGame called CS228BlockGame that implements the game described in the introduction. The score should be the total number of individual blocks that have been collapsed

in the game so far. The methods determineCellsToCollapse and determineScore must be declared **public** (this requirement is to make it easier for us to test your code).

The initial state of the grid is as follows: in the bottom 8 rows, alternating locations are filled with a randomly chosen block in a checkerboard pattern, as illustrated in the following screenshot (which also depicts a falling ITriomino).



The BasicGenerator class

See the javadoc for the IPolyominoGenerator interface.

You'll need to implement a BasicGenerator class implementing IPolyominoGenerator so that it returns one of the 3 concrete polyomino classes, chosen at random with the following weighting:

ITriomino 60%
LTetromino 20%
SlashDomino 20%

In all cases the colors of the blocks should be assigned uniformly at random from the array AbstractBlockClass.COLORS. Duplicate colors are allowed in a polyomino.

The starting position of each polyomino overlaps the first row of the grid (but not the second row) and is (approximately) centered left-to-right within the grid. Specifically you must use the following initial positions for the bounding square for each type:

ITriomino (5, -2)
LTetromino (5, -2)
SlashDomino (5, -1)

Getting started

The user interface, and other source code, in the cs228hw1.zip archive is an Eclipse project that you can import and build. Run the main class edu.iastate.cs228.hw1.ui.GameMain. This will start up a SampleGame and you should see a simple animation which shows a "Sampleomino" (a simple three-cell Polyomino) falling from top to bottom. You can put it into "fast drop" mode by pressing the down arrow key.

A good place to start is to implement the missing methods of Sampleomino and AbstractBlockGame that will enable you to move the Sampleomino left and right, which will allow you to play a rudimentary game of Tetris.

JUnit

See the "Unit Testing" section of the Web Links on WebCT for JUnit resources and examples.

Documentation and style

Roughly 15% of the points will be for documentation and style. There is a brief set of guidelines to follow in the "Course Specific" section of the Web Links on our WebCT page.

What to turn in

Detailed submission instructions will be provided in the clarification thread.

Late Penalties

Assignments may be submitted up to 24 hours after the deadline with a 25% penalty (not counting weekends/holidays). No credit will be given for assignments submitted after that time.