Tetris (18 Punkte + 2 Bonuspunkte)

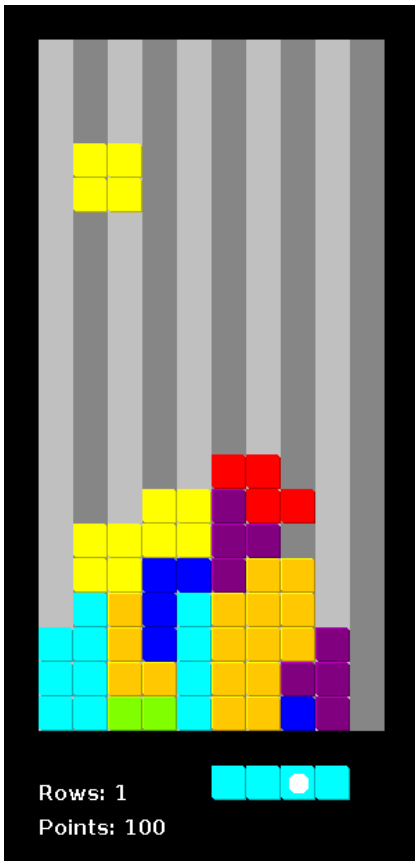
In diesem Projekt ist es Ihre Aufgabe das Spiel Tetris zu programmieren. Beachten Sie, dass Sie alle public Tests bestehen müssen, um Punkte für das Projekt zu bekommen.

1 Spielprinzip

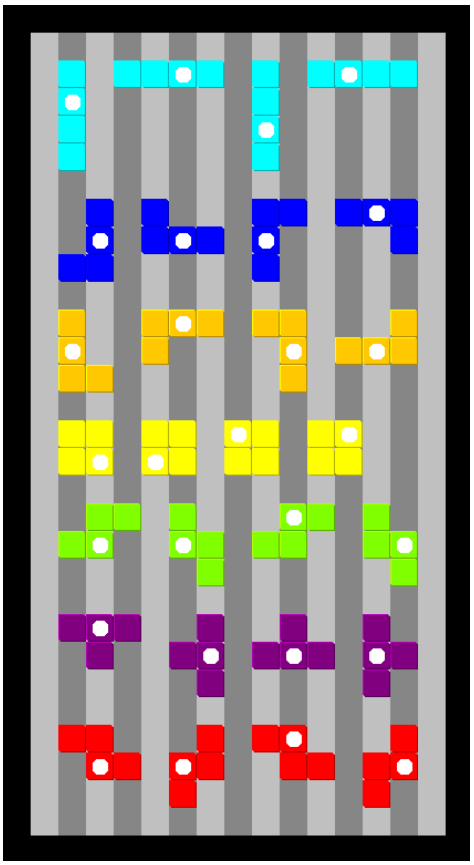
Das Spiel findet auf einem gitterartigen Spielfeld (`Board`) statt. Während des Spiels fallen verschiedene Spielsteine (`Piece`) herab. Ziel ist es, diese Steine durch Rotation und horizontales Verschieben so anzuordnen, dass möglichst lückenlose Reihen entstehen. Um sich das Spielprinzip zu verdeutlichen, können Sie eine Beschreibung oder eine der zahlreichen, frei verfügbaren Implementierungen im Internet nutzen[1].

2 Aufgabe

Ihre Aufgabe ist es, die Spielfunktionalität zu implementieren. Dies beinhaltet die Spielsteine (`Piece`), das Spielfeld (`Board`) und das Spiel selbst (`TetrisGame`). Des Weiteren soll ein Autoplayer implementiert werden der in der Lage ist selbständig Tetris zu spielen.



(a) Tetris Spielansicht (`TetrisComponent`)



(b) Tetris Spielsteinansicht (`PieceComponent`)

Abbildung 1: Verschiedene mitgelieferten Tetrisansichten.

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

3 Spielstein

Ein Spielstein (`Piece`) besteht aus vier zusammenhängenden Feldern. Es gibt sieben verschiedene Grundformen (`I, J, L, O, S, T, Z`). Der Name der Spielsteine richtet sich nach der Form des Buchstaben. Abbildung 1b zeigt die verschiedenen Spielsteine. Die Spielsteine werden von einer Spielsteinfabrik (`PieceFactory`) erzeugt. Ein Spielstein hat folgende Eigenschaften und Konzepte:

Körper Jeder Stein wird durch ein zweidimensionales *boolean* Array beschrieben, wobei zuerst die Reihe, dann die Spalte adressiert wird. Der Punkt $(0, 0)$ befindet sich oben links. Eine Position hat den Wert *true*, wenn diese Stelle vom Spielstein belegt ist. Das Array muss immer genau so hoch und breit sein, wie es notwendig ist, um den Spielstein aufzunehmen.

Rotationspunkt Jeder Spielstein hat einen Rotationspunkt. Er gibt an, um welche Stelle ein Stein rotiert wird. Wenn ein Stein auf dem Spielfeld platziert wird, werden immer die Koordinaten angegeben, an denen der Rotationspunkt platziert werden soll. In Abbildung 1b sind die Rotationspunkte farblich hervorgehoben.

Rotation Jeder Spielstein kann um den Rotationspunkt rotiert werden. Dies kann im Uhrzeigersinn (`getClockwiseRotation()`) und gegen den Uhrzeigersinn (`getCounterClockwiseRotation()`) geschehen.

Nach vier Rotationen kommt jeder Spielstein wieder in seine Ausgangslage. Hier soll sichergestellt werden, dass die `equals(Object obj)` Methode (siehe `java.lang.Object`) *true* zurückliefert, wenn dies der Fall ist, also der gleiche Körper in der gleichen Lage ist und der Rotationspunkt übereinstimmt.

Typ Jeder Spielstein hat einen Typ (`PieceType`), der angibt, aus welcher Grundform dieser Stein durch Rotation entstanden ist.

Kopierbarkeit Ihre Implementierung sollte die Methode `clone` bereitstellen, die eine tiefe Kopie des Spielsteins zurückgibt. Hierbei ist darauf zu achten, dass eine tiefe Kopie des zweidimensionalen Arrays gemacht wird, welches den Spielstein repräsentiert. Das bedeutet, dass jeder Wert kopiert werden muss und nicht nur die Referenz auf das Array.

4 Spielfeld

Das Spielfeld (`Board`) ist ein zweidimensionales Gitter, auf dem die Spielsteine platziert und entfernt werden können. Hier gilt, wie bei den Spielsteinen: Zuerst wird die Reihe dann die Spalte adressiert, und $(0, 0)$ bezeichnet die linke obere Ecke.
Abbildung 1b zeigt das von `PieceComponent` erzeugte Bild. Sobald Sie

- die Klassen `Piece` und `PieceFactory`,
- die Methoden `getBoard()` und `getNumberOfRows/-Columns()` des Spielfeldes,
- und `createPieceFactory()` und `createBoard()` in `MyTetrisFactory`

implementiert haben, können Sie ihre Implementierung mit Hilfe der Abbildung überprüfen. Führen Sie dazu die `main` Funktion in der `PieceComponent` als Java Anwendung aus [2].

Spielstein platzieren Ein Spielstein wird auf dem Spielfeld platziert, indem ein Spielstein sowie die Position (Reihe, Spalte) angegeben werden (wobei die Position die des Rotationspunkts ist). Die Methode `addPiece()` platziert einen Stein auf dem Spielfeld oder wirft eine `IllegalArgumentException`, wenn dies nicht möglich ist. Mit der Methode `canAddPiece()` kann überprüft werden, ob der Stein auf dem Spielfeld an der gewünschten Position platziert werden kann.

Spielstein entfernen Das Entfernen der Spielsteine verhält sich analog zum Platzieren der Steine. Die Methode `removePiece()` entfernt einen Stein. Mit `canRemovePiece()` kann überprüft werden, ob es möglich ist, einen Stein zu entfernen.

Vollständige Reihen In der Methode `deleteCompleteRows()` wird geprüft, ob vollständige Reihen auf dem Spielfeld vorhanden sind. Ist dies der Fall, werden die Reihen vom Spielfeld entfernt und alle drüber liegenden Reihen bewegen sich als Ganzes nach unten.

[2] In Eclipse können Sie dies mit einem Rechtsklick auf die Datei `Runs as` → `Java Application` tun.

Kopierbarkeit Ihre Implementierung sollte die Methode `clone` bereitstellen, die eine tiefe Kopie des Spielfelds zurückgibt (wie bereits bei dem Spielstein gefordert).

5 Das Spiel

Das Interface `TetrisGame` repräsentiert das Tetris Spiel. Eine Implementierung dieser Klasse sollte unter anderem folgende Felder verwalten:

Board Das Spielfeld auf dem gespielt wird. Initial ist das Spielfeld leer und erst nach einem Aufruf von `newPiece()` oder `step()` gibt es einen aktuellen Spielstein (`currentPiece`). Falls `step()` auf einem noch leeren Spielfeld ohne aktuellen Spielstein aufgerufen wird soll sich die Funktion genau wie `newPiece()` verhalten. Sollte das Platzieren des ersten Spielsteins nicht möglich sein, ist das Verhalten undefiniert.

Spielsteine und Position Den aktuellen Spielstein (`currentPiece`), dessen Position (`pieceRow` und `pieceColumn`), sowie den nächsten Spielstein (`nextPiece`). Initial soll das Spiel bereits den nächsten Stein bereithalten.

Reihen und Punkte Die im Spiel erreichten Punkte und komplette Reihen. Hierbei wird folgendes Punkteschema benutzt: Für eine komplette Reihe gibt es 100 Punkte, für zwei 300, für drei 500 und für vier 1000 Punkte.

Außerdem stellt das Interface unter anderem folgende Funktionen bereit:

Bewegen des Spielsteins Der aktuelle Spielstein kann um ein Feld nach rechts, links oder unten bewegt werden, sofern dies möglich ist.

Rotieren des Spielsteins Der aktuelle Spielstein kann rotiert werden. Dies ist genau dann möglich, wenn die Zielposition der Rotation nicht belegt ist. Insbesondere müssen Hindernisse auf dem Weg nicht beachtet werden.

Neuer Spielstein Die Methode `newPiece()` bringt einen neuen Spielstein (`nextPiece`) ins Spiel und bestimmt einen zufälligen nächsten Spielstein. Die Position des neuen Spielsteins ist

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

Bevor der Spielstein auf dem Spielfeld platziert wird, wird überprüft, ob komplette Reihen auf dem Spielfeld sind, und wenn dies der Fall ist, werden die entsprechenden Methoden aufgerufen. Kann der neue Spielstein danach nicht auf dem Feld platziert werden, ist das Spiel vorbei (Game Over).

Spielschritt Die Funktion `step()` bewegt den aktuellen Stein um eine Reihe nach unten. Wenn dies nicht möglich ist, wird ein neuer Spielstein mit `newPiece()` ins Spiel gebracht.

Sie können die Klasse `BoardComponent` bzw. die Methode `createBoardView()` in der Klasse `Views` verwenden um Ihre Implementierung des Spielfelds zu testen. Erstere zeigt Ihnen ein leeres Spielfeld an und zweiteres das als Argument übergebene Spielfeld (siehe Abbildung 1a).

5.1 Observer

Das Interface `TetrisGame` erweitert das Interface `GameObservable`. Ihre Implementierung muss deshalb alle angemeldeten `GameObserver` bei folgenden Ereignissen informieren.

Veränderung der Position Wurde die Position oder Rotation des aktuellen Spielsteins erfolgreich verändert, wird die Methode `piecePositionChanged()` der angemeldeten Observer aufgerufen. Dies geschieht nach jeder Änderung, d.h. wird ein Stein um zwei Felder nach links bewegt, wird die Methode auch zweimal aufgerufen.

Spielstein landet Wenn der aktuelle Spielstein gelandet ist, d.h. er kann nicht mehr weiter nach unten bewegt werden, wird die Methode `pieceLanded()` der Observer aufgerufen. Dies geschieht in der Methode `step()`, bevor auf komplette Reihen überprüft und der neue Spielstein auf dem Spielfeld platziert worden ist. Der aktuelle Spielstein ist zu diesem Zeitpunkt noch nicht aktualisiert.

Komplette Reihen Wenn durch das Landen eines Spielsteins komplette Reihen entstanden sind, wird die Methode `rowsCompleted()` der Observer aufgerufen.

Ende des Spiels Wenn das Spiel beendet ist, d.h. kein neuer Stein in der Methode `step()` platziert werden kann, wird die Methode `gameOver()` der Observer aufgerufen.

6 Autoplayer

Ein Autoplayer kann automatisch Tetris spielen. Dazu implementiert er das `GameObserver` Interface, um über Veränderungen am Spiel informiert zu werden sowie das `AutoPlayer` Interface. Solange das Spiel nicht beendet ist wird mit der Methode `getMove()` ein Zug von dem Autoplayer angefragt. Die möglichen Züge sind im Enum `AutoPlayer.Move` aufgelistet. In diesem Modus entscheidet der Autoplayer selbst wann ein `step` ausgeführt wird in dem er `Move.DOWN` als nächsten Zug auswählt. Die Klasse `AutoplayerView` startet das Spiel. Es wird solange `getMove()` aufgerufen und der Zug ausgeführt bis das Spiel beendet ist. Falls ein Zug nicht möglich ist wird das Spiel automatisch beendet. Der Autoplayer darf nur über die von uns vorgegebenen Interfaces und auch nur lesend auf den Spielzustand zugreifen.

6.1 Bonus: Schlagen Sie die Tutoren (2 Bonuspunkte)

Sie erhalten zwei Bonuspunkte, wenn Sie den Autoplayer unserer Tutoren schlagen. Bei dem Duell werden wir Ihren Autoplayer gegen den der Tutoren auf den gleichen Random-Seeds antreten lassen.

6.2 Autoplayer-Turnier

Am Ende der Vorlesung wird ein Turnier stattfinden, in dem Ihr `AutoPlayer` gegen die Implementierungen Ihrer Kommilitonen antritt. Es wird ein gesondertes Dokument geben, in dem die Turnierregeln klargestellt werden. Bitte beachten Sie, dass die Teilnahme am Turnier freiwillig ist und keine Zusatzpunkte einbringt.

7 Tetris spielen

Sie können ihr Tetris selbst spielen indem Sie die Datei `PlayerView.java` als Javaanwendung ausführen. Mit den Pfeiltasten *links*, *rechts* und *runter* können Sie die Steine bewegen und mit *w* und *q* im bzw. gegen den Uhrzeigersinn drehen. Das Spiel funktioniert sobald Sie alles bis auf den AutoPlayer implementiert haben.

8 Eigene Tests

Sie können eigene Tests in das `tetris.tests` Paket legen um sie gegen die Referenzimplementierung zu testen. Eigene Tests gehen nicht in die Bewertung ein.

Projekt aufsetzen

Um das Projekt in Eclipse bearbeiten zu können, müssen Sie erst das Depot auschecken und das Projekt importieren.

1. Klonen Sie das Projekt in einen beliebigen Ordner:
   `git clone https://prog2scm.cdl.uni-saarland.de/git/project5/$NAME /home/prog2/project5`
   wobei Sie $NAME durch ihren CMS-Benutzernamen ersetzen müssen.
2. Benutzen Sie *Import*, ein Unterpunkt des *File* Menüeintrags in Eclipse, um den Importierdialog zu öffnen.
3. Wählen Sie „Existing project into workspace“ aus und benutzen Sie den neuen Dialog um den Ordner des Projekts (siehe Punkt 1) auszuwählen.

Hinweise

- Wir verwenden Fabrikmethoden der Klasse `MyTetrisFactory`, um Ihre Implementierung zu testen.
- Ändern Sie die existierenden Dateien in dem Projekt nicht ab, da wir sie vor dem Testen auf den Ursprungszustand zurücksetzen. Dies gilt nicht für `MyTetrisTest.java` und `MyTetrisFactory.java`, welche Sie für Ihre Tests beziehungsweise die Implementierung nutzen können. Sie dürfen neue Dateien in den Paketen anlegen, die mit `tetris.` beginnen.

Viel Erfolg!