UNIVERSITÄT
DES
SAARLANDES

Programmierung 2 - SS19

Projekt 5 - Tetris

Autoren: Jonas Klesen, Emma Hoffmann

19 Juni 2019

Universität des Saarlandes

Überblick

1. Technische Hinweise
2. Allgemeines
3. Struktur des Projektes
4. Autoplayer

Technische Hinweise

Git Projekt-Repository

Wir können das Projekt mit `git clone` unter folgender URL beziehen:

```
https://prog2scm.cdl.uni-saarland.de/git/project5/<NAME>
```

<NAME> = Euer Benutzername auf der Prog2-Website

Achtung!

Die Repositories sind nur innerhalb des Uninetzes erreichbar. Von außerhalb kann man eine VPN-Verbindung zum Uninetz einrichten.

Eine Anleitung steht auf der Website unter Software.

Import in Eclipse

1. Projekt klonen
2. Projekt importieren:
    - Unterpunkt *Import* des Menüeintrags *File*

Import in Eclipse

1. Projekt klonen
2. Projekt importieren:
   - Unterpunkt *Import* des Menüeintrags *File*
   - *General* und dann *Existing project into workspace* auswählen

Import in Eclipse

1. Projekt klonen
2. Projekt importieren:
   - Unterpunkt *Import* des Menüeintrags *File*
   - *General* und dann *Existing project into workspace* auswählen
   - Geklonten Ordner auswählen

Allgemeines

Enums

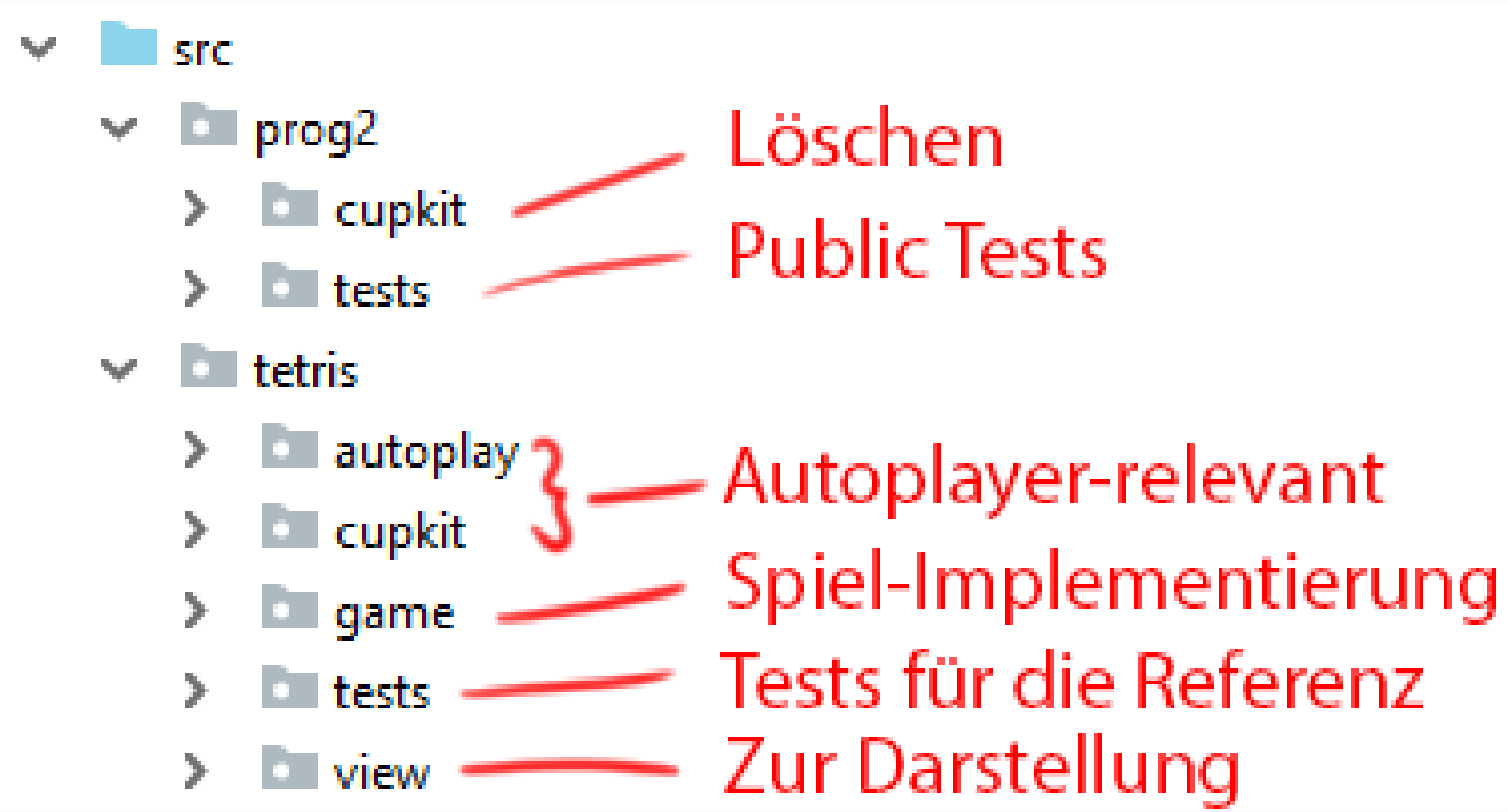
Zum Beispiel:

```
1     enum Geschmack {
2         BITTER,
3         SUESS,
4         SAUER,
5         SALZIG,
6         UMAMI
7     }
8
9     Geschmack g = Geschmack.SAUER;
```

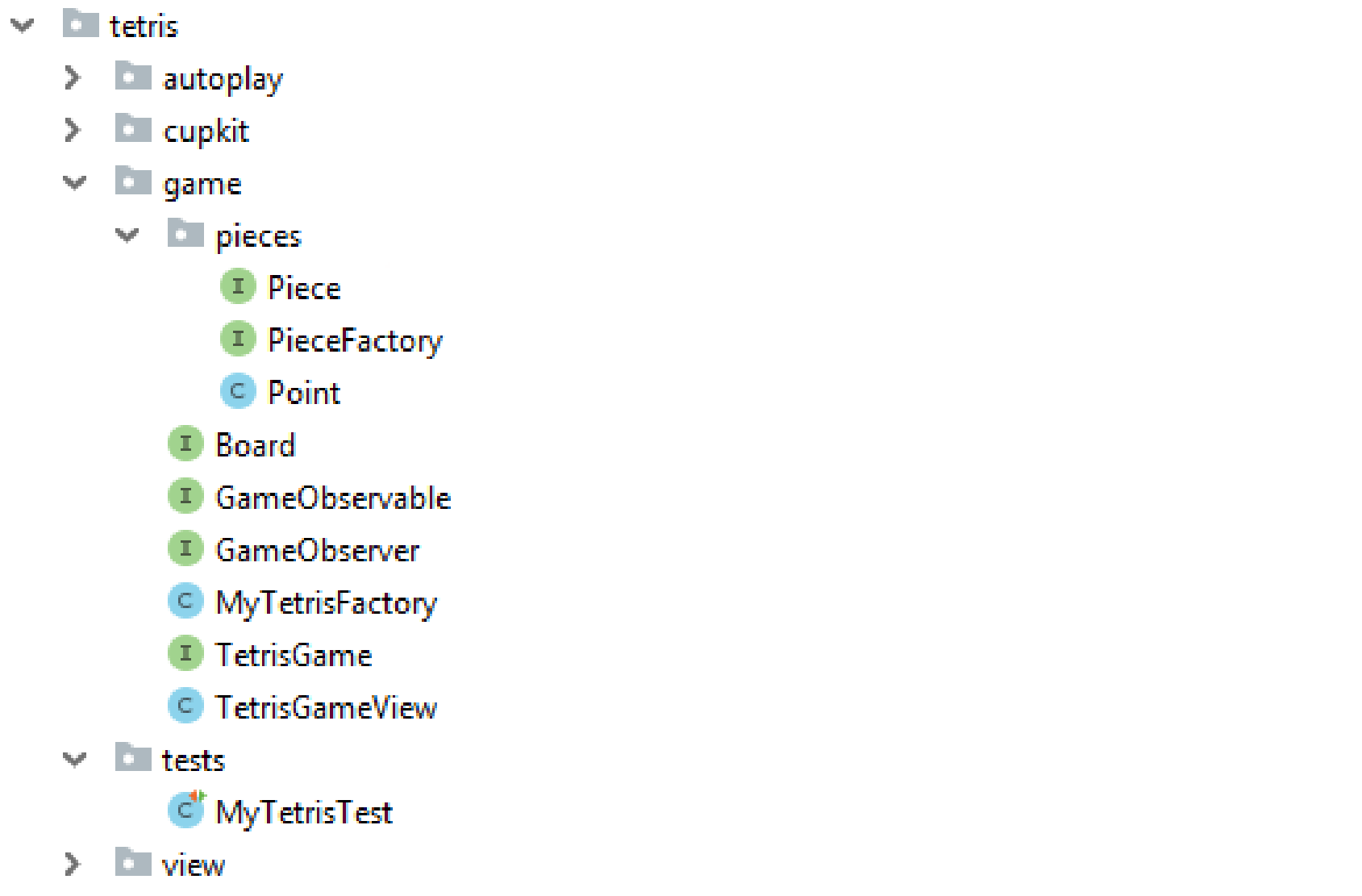
$\tilde{=}$ Spezielle Art von Klasse, die nur Konstanten enthält.

Struktur des Projektes

Übersicht - Ordner src

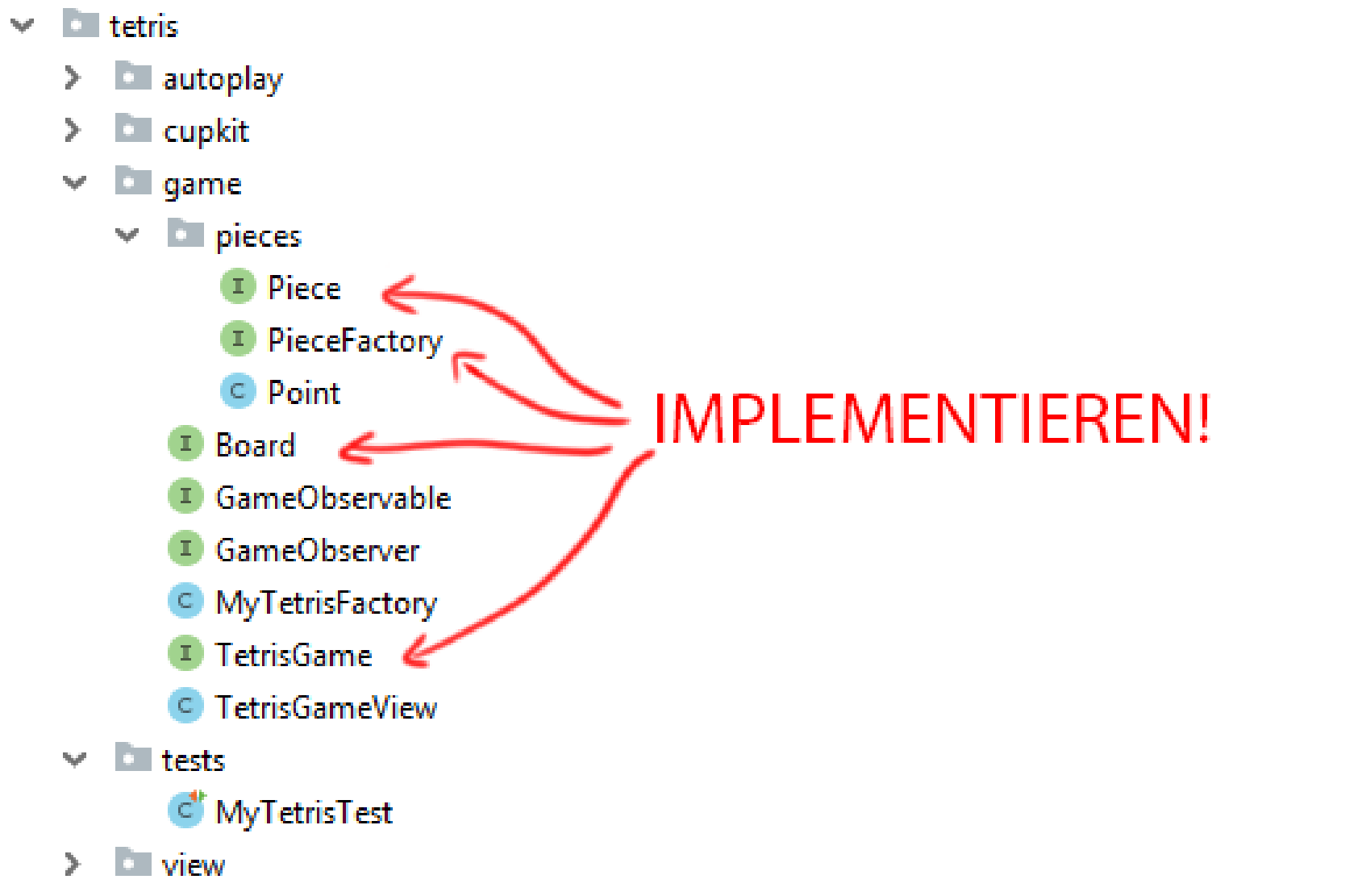


Übersicht - Package game



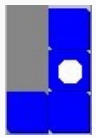
Übersicht - Package game

Übersicht - Package game

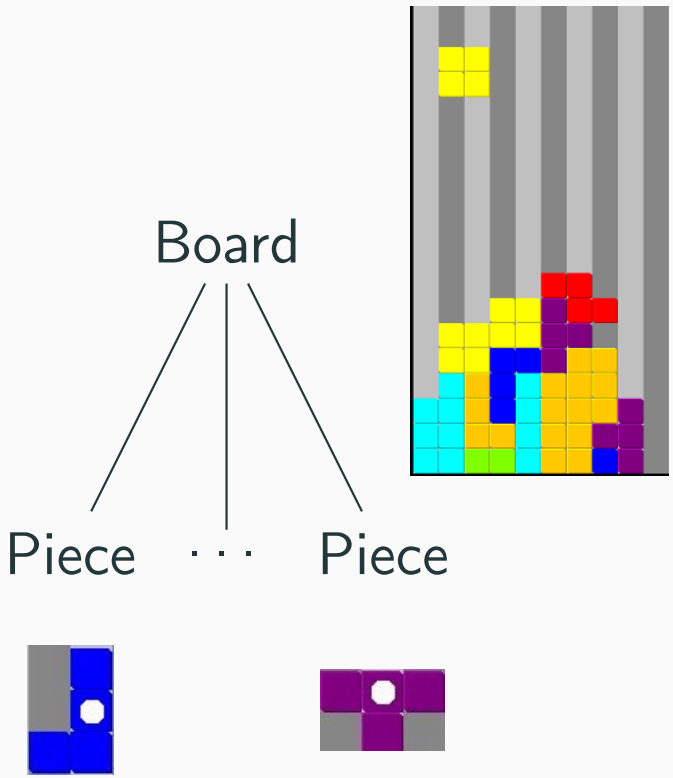


Übersicht

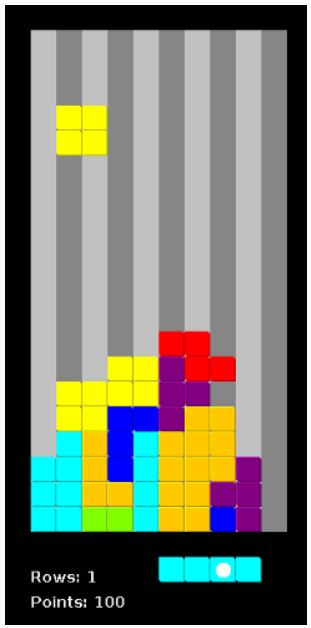
Piece $\cdots$ Piece

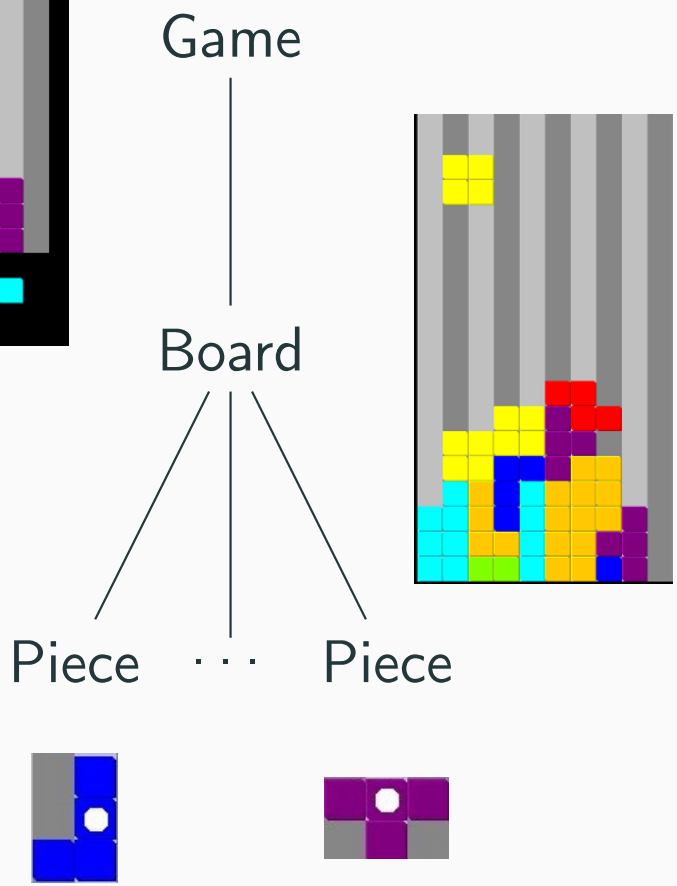


Übersicht

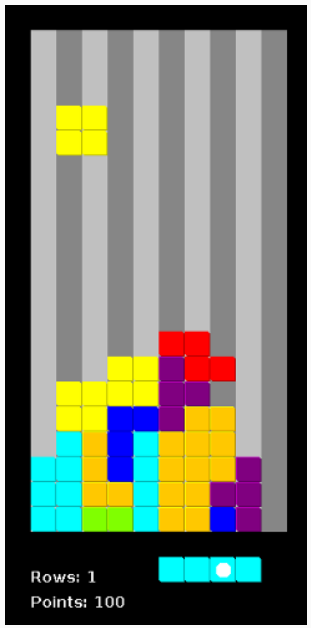


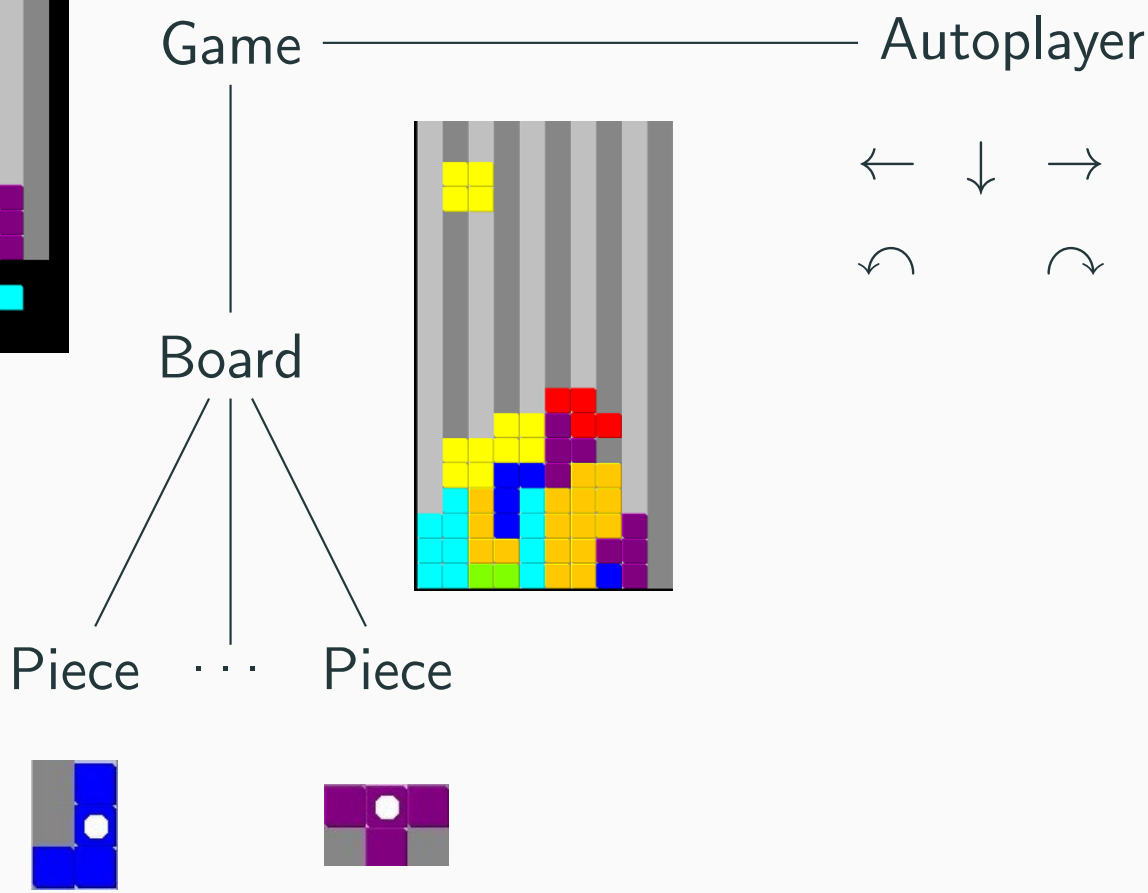
Übersicht





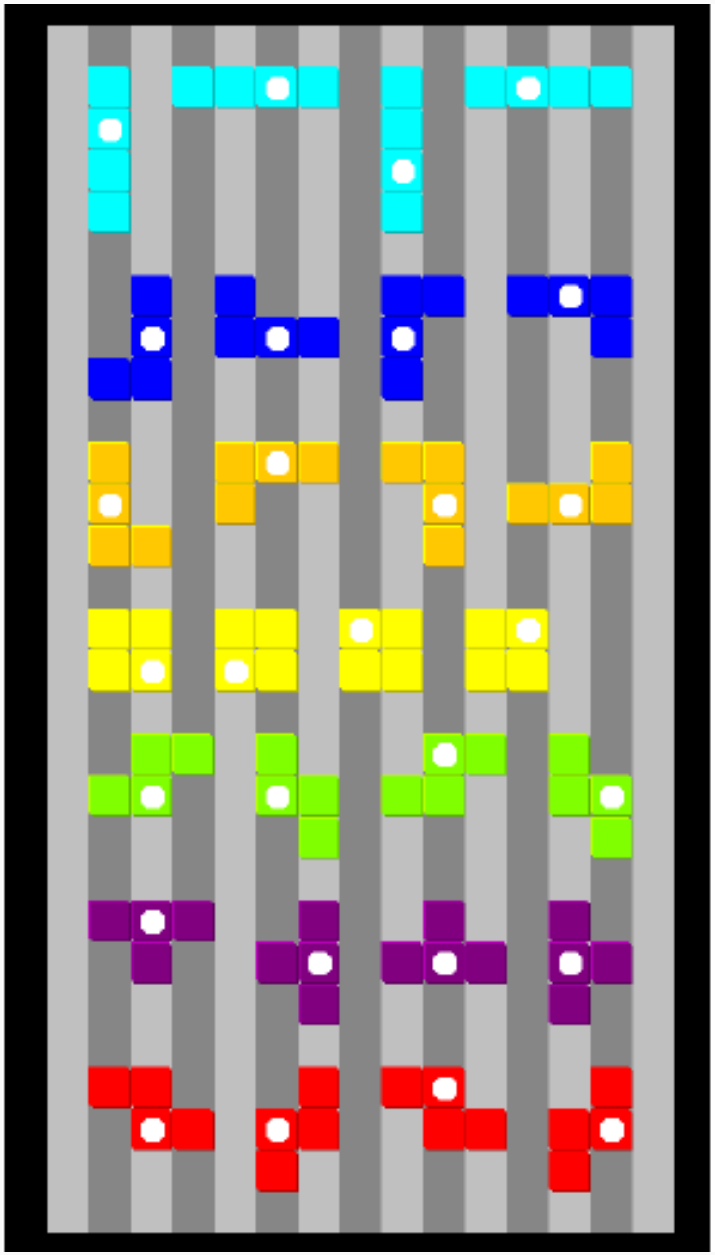
Übersicht





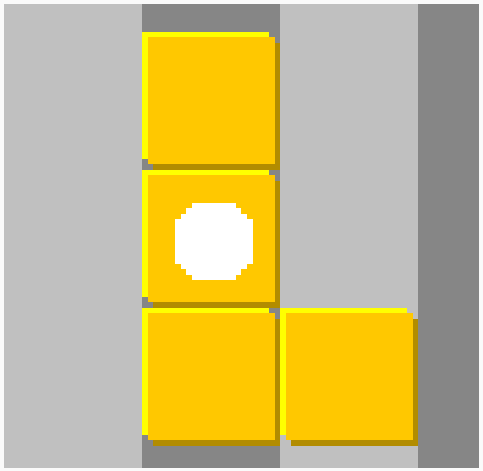
Pieces

Pieces - Übersicht



Pieces - interne Struktur

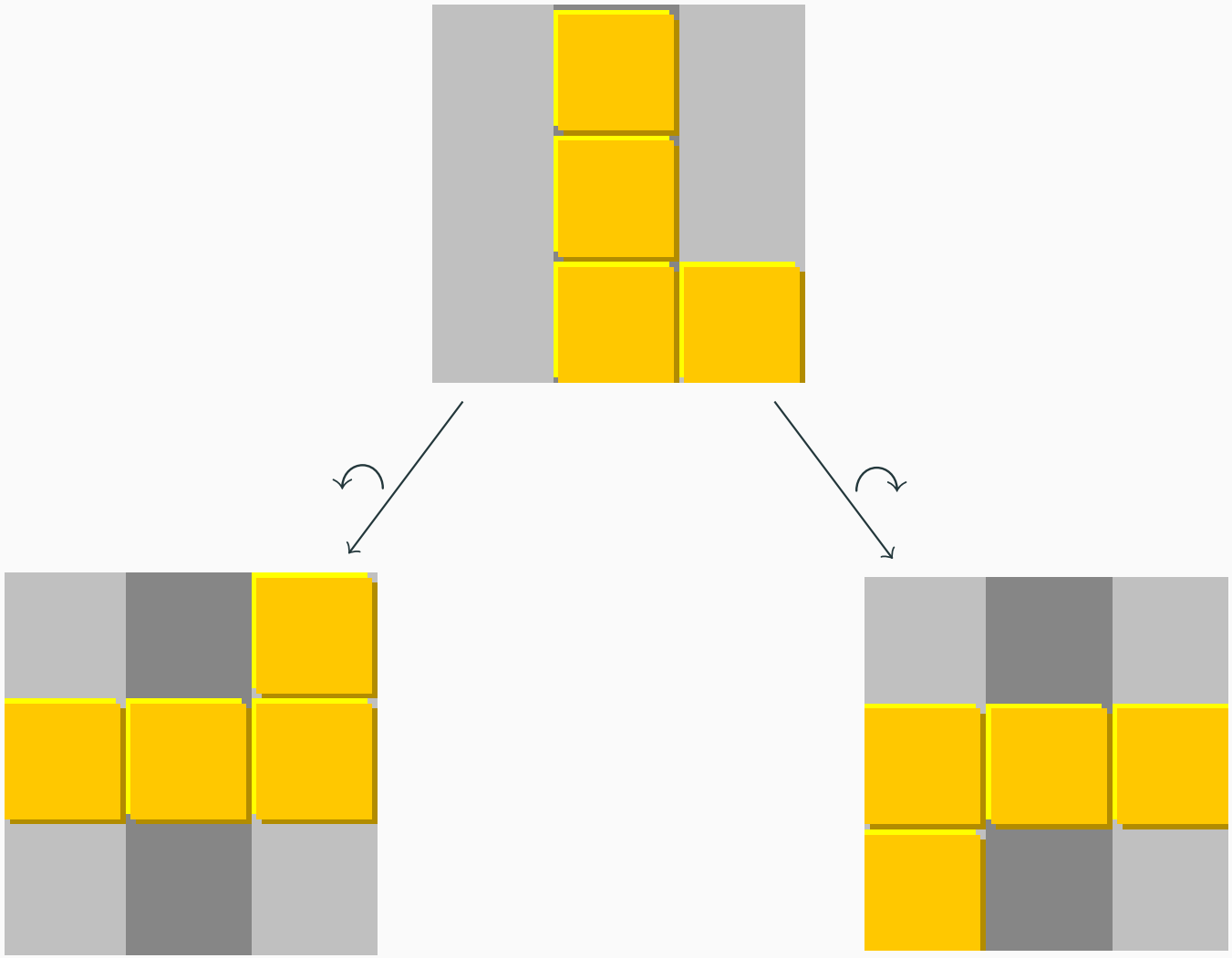
Beispiel: 'L'-Piece



`true`	`false`
`true`	`false`
`true`	`true`

Rotationspunkt(weiß): Point(1,0) -> erst Reihe, dann Spalte

Pieces - Rotation



Array-Dimension: (2x3)

Array-Dimension: (2x3)

Pieces - Das enum PieceType

```
1 enum PieceType {
2     L, J, T, O, I, Z, S;
3 }
```

In der Implementierung dann, z.Bsp.:

```
1 private PieceType type;
2 ...
3 type = PieceType.L;
```

Achtung! Das Interface `Piece` darf **nicht** verändert werden!

→ Denkt über eine sinnvolle Klassenhierarchie nach.

Piece Interface - Besonderheiten

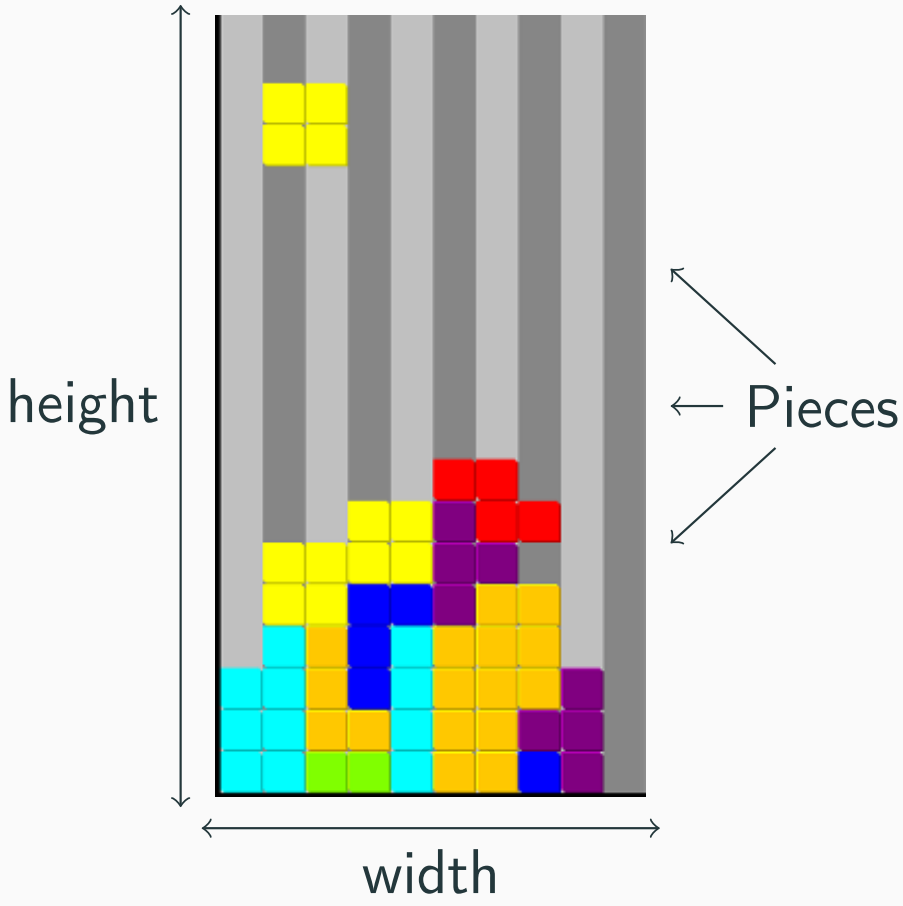
- `Piece getClockwiseRotation()`
  → Verändert Piece, auf dem sie aufgerufen wurde **nicht**!

- `Piece getCounterClockwiseRotation()`
  →Verändert Piece, auf dem sie aufgerufen wurde **nicht**!

- `boolean equals(Object other)`
  → Wenn PieceType, Rotation, Rotationspunkt übereinstimmen

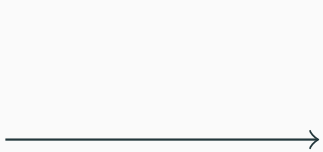
Das Board

Board



Board - Spielfeld

Zweidimensionales Array, vom Typ `PieceType`



Z	Z	null
T	Z	Z
T	T	null

Board Interface - Besonderheiten

- `void addPiece(Piece piece, int row, int column)`
  → `row` und `column`: Hier landet der Rotationspunkt
- `boolean canAddPiece(Piece piece, int row, int column)`
  → Wenn alle Felder die der Stein dort braucht frei sind

- `removePiece`, `canRemovePiece` ähnlich

- `int deleteCompleteRows()`
  → Sucht und entfernt alle vollen Zeilen, gibt Anzahl zurück

Fragen?

Das Game

Game - Das muss es sich merken

- Board
- aktueller Spielstein und dessen Position (Rotationspunkt)
- nächster Spielstein
- Anzahl gelöschter Zeilen und Punktestand
- Game over?
- Liste an 'Zuschauern' (Observern)
  $\rightarrow$ Wollen über Geschehnisse informiert werden

Game - aktueller Spielstein

- Spielstein bewegen (links, rechts, unten) - wenn möglich!
- Spielstein rotieren (links, rechts) - wenn möglich!
  $\rightarrow$ Rotationspunkt bleibt an gleicher Stelle!
- boolean newPiece() $\Rightarrow$ Neuer Stein platzieren, wenn möglich

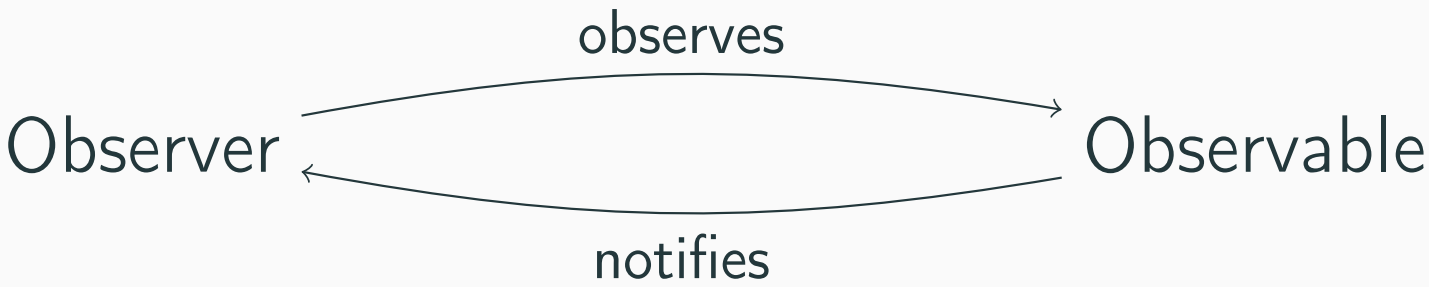
Game - Spielschritt

1. Aktuellen Spielstein nach unten bewegen

2. Nicht möglich $\rightarrow$ nächsten Spielstein setzen

3. Auch das nicht möglich $\rightarrow$ Game over!

Beobachtermuster (Observer Pattern)



void tellX(Observable o)

void tellY(Observable o)

⋮

void register(Observer o)

void unregister(Observer o)

In unserem Fall: GameObserver

Er registriert sich bei eurer Implementierung vom Interface `TetrisGame` mittels `addObserver`, `removeObserver`.

Möchte informiert werden wenn...

- Einer oder mehrere Zeilen entfernt wurde
- Die Position des aktuellen Steins sich geändert hat
  $\rightarrow$ Auch bei Rotation!
- Ein Stein unten angekommen ist
- Das Spiel vorbei ist

Achtung: Es können sich mehrere Observer registrieren!

Sonstiges

Nicht klar, was Funktion tun soll?

Kommentare in den Interfaces beachten!

Euer Programm testen

Im Package prog2.tetris.view gibt es...

- **PieceComponent:** Zeigt alle Steine in allen Rotationen an
  $\rightarrow$ Sollte aussehen wie Abbildung 1(b)
- **BoardComponent:** Zeigt ein leeres Board an
- **PlayerView :** Startet das Spiel

Tipp: Man kann diese Programme auch im Debug-Modus starten!
$\rightarrow$ Mit Breakpoints an der richtigen Stelle findet man schnell Fehler

Fragen?

Autoplayer

Autoplayer - Interface

Achtung: Es gibt Daily-Tests zum Autoplayer!

Das Interface Autoplayer:

```
1 public interface AutoPlayer extends GameObserver{
2 public enum Move {
3   RIGHT, LEFT, DOWN, ROTATE_CW, ROTATE_CCW,
4 }
5
6 Move getMove(); // <-- Funktion
7 }
```

Eure Implementierung davon braucht noch:

- Einen Konstruktor, der ein `TetrisGameView` annimmt.
- Die Methoden aus GameObserver

Autoplayer - Vorgehen

- Zugriff auf Zustand via TetrisGameView
  $\rightarrow$ `getBoardCopy`, `getCurrentPieceCopy`,...
- Berechnung des nächsten Zuges in getMove():
  - Berechnung der möglichen Endzustände (≤ 40)
  - Bewertung aller Endzustände
    - nicht geschlossene Lücken
    - gelöschte Reihen
    - Vermeidung eines GameOvers
  - Führe den bestmöglichen Zug aus

Fragen?