README.md

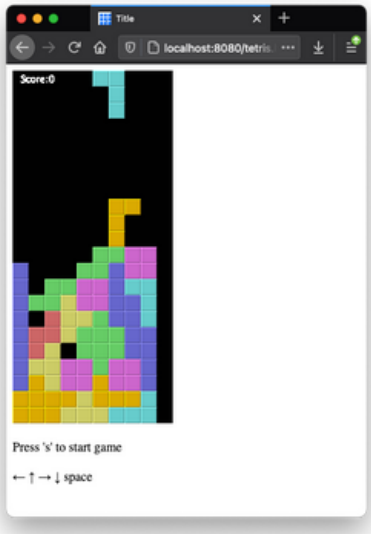
WebSocket example

This is a WebSocket example project.

Usage

```
$ ./gradlew run
```

Build is executed, the browser will be launched, and the application will start.



はじめに

前回からの続きです。

blog1.mammb.com

前回は Java API for WebSocket について見てきました。 今回は、前回の内容を踏まえ、テトリスの実装を行っていきます。

ここで使用するコードは以下に置いてあります。

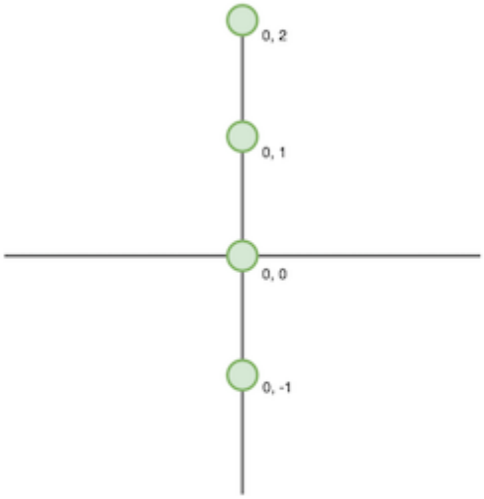
github.com

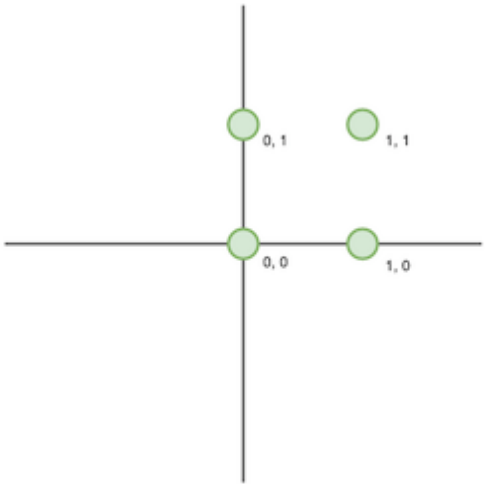
テトロミノ

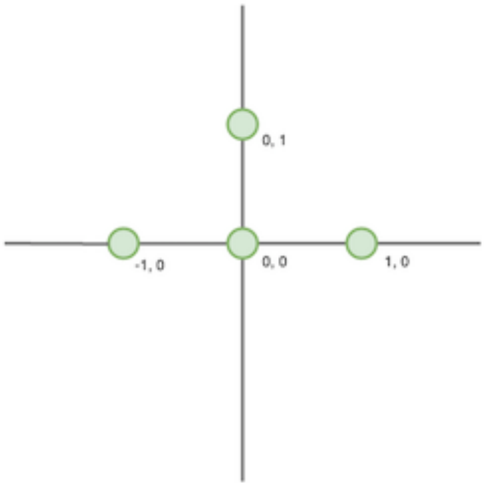
同じ大きさの4個の正方形を辺に沿ってつなげた形を総称してテトロミノと呼びます。 4つの正方形を辺に沿ってつなげた形は回転によって同じになるものを同一と考えると7種類があります。

それぞれを I型、O型、T型、L型、S型、L型の鏡像をJ型、S型の鏡像をZ型とします。

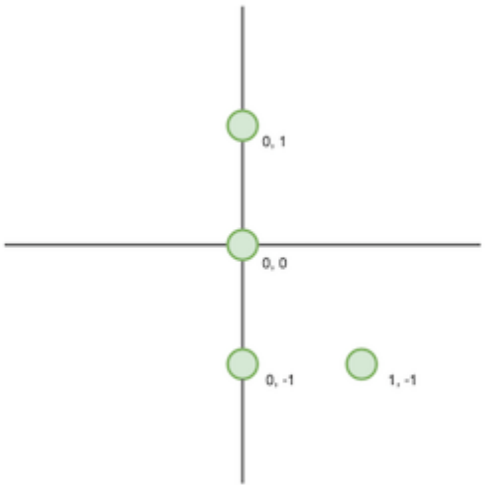
I型、O型、T型 は以下のような形です。

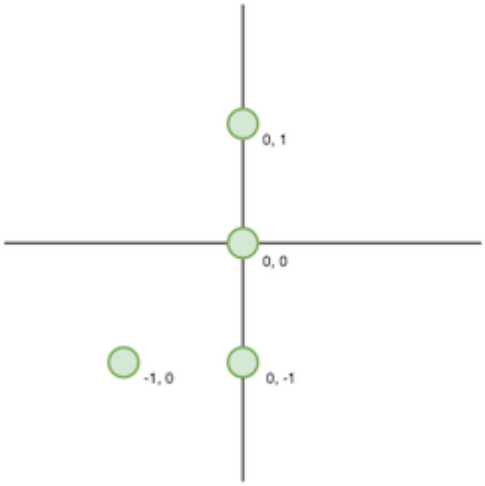




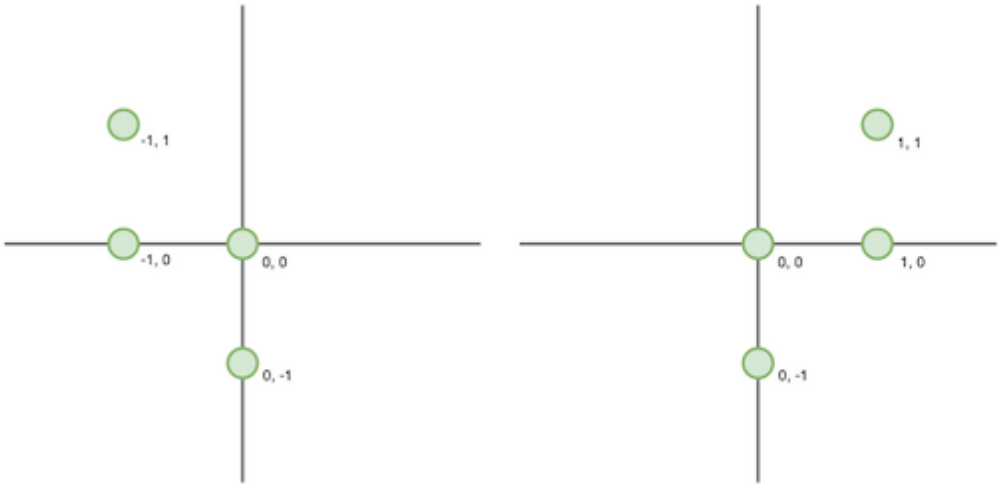


L型、J型は以下のような形です。





S型、Z型は以下のような形です。



これらの型を列挙で以下のように定義します。

```
public enum Tetrominoe {
    S(new int[][] { {0, -1}, {0, 0}, {-1, 0}, {-1, 1} },
        new Color(204, 102, 102)),
    Z(new int[][] { {0, -1}, {0, 0}, {1, 0}, {1, 1} },
        new Color(102, 204, 102)),
    I(new int[][] { {0, -1}, {0, 0}, {0, 1}, {0, 2} },
        new Color(102, 102, 204)),
    T(new int[][] { {-1, 0}, {0, 0}, {1, 0}, {0, 1} },
        new Color(204, 204, 102)),
    O(new int[][] { {0, 0}, {1, 0}, {0, 1}, {1, 1} },
        new Color(204, 102, 204)),
    J(new int[][] { {-1, -1}, {0, -1}, {0, 0}, {0, 1} },
        new Color(102, 204, 204)),
    L(new int[][] { {1, -1}, {0, -1}, {0, 0}, {0, 1} },
        new Color(218, 170,   0)),
    X(new int[][] { {0, 0}, {0, 0}, {0, 0}, {0, 0} },
        new Color(0, 0, 0)),
    ;

    private final int[][] points;
    private final Color color;

    Tetrominoe(int[][] points, Color color) {
        this.points = points;
        this.color = color;
```

```
    }
}
```

後の扱いを簡便にするために、空の型を表す X という定義を追加しています。

ランダムなテトロミノを得るためのスタティックメソッドを定義しておきましょう。

```
public enum Tetrominoe {
    // ...
    private static final SplittableRandom random = new SplittableRandom();
    public static Tetrominoe random() {
        return Tetrominoe.values()[Math.abs(random.nextInt()) % 7];
    }
}
```

加えて、O型は回転しても変わらないため、回転可能かを判断するヘルパーメソッドと、テトリミノの座標を取得するメソッドを定義しておきます。

```
    public boolean isRotatable() {
        return this != O && this != X;
    }

    public int[][] getPoints() {
        var ret = new int[4][2];
        for (int i = 0; i < 4 ; i++) {
            for (int j = 0; j < 2; ++j) {
                ret[i][j] = points[i][j];
            }
        }
        return ret;
    }
```

ゲーム盤面上のブロック

ゲーム盤面上で扱うブロックを定義します。

```
public class Block {

    public static final Block empty = new Block(Tetrominoe.X);

    private Tetrominoe type;
    private final int points[][];

    private Block(Tetrominoe type) {
        this.type = type;
        this.points = type.getPoints();
    }

    public static Block randomOf() {
        return new Block(Tetrominoe.random());
    }
}
```

このブロックは回転させる必要があるため、回転用のメソッドを定義しましょう。

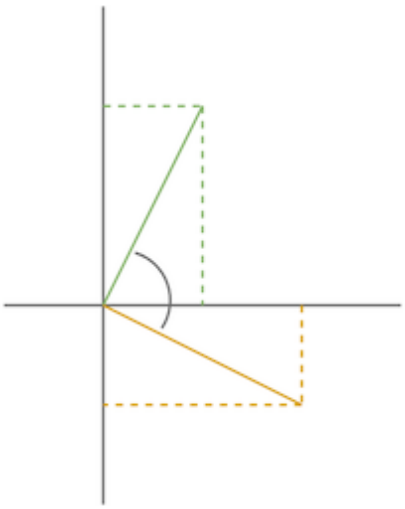
以下のようになります。

```
    public Block rotateLeft() {
        if (!type.isRotatable()) {
            return this;
        }
        var result = new Block(type);
        for (int i = 0; i < 4; ++i) {
            result.setX(i, y(i));
            result.setY(i, -x(i));
        }
        return result;
    }

    public Block rotateRight() {
        if (!type.isRotatable()) {
```

```
            return this;
        }

        var result = new Block(type);
        for (int i = 0; i < 4; ++i) {
            result.setX(i, -y(i));
            result.setY(i, x(i));
        }

        return result;
    }
```

原点から90度単位での回転だけ考えれば良いため、座標の x と y を入れ替えて一方の符号を反転させれば完了です。

以下のような図を考えると分かりやすいかもしれません。



ゲームの実装

```
public class Tetris {

    private final int UNIT_SIZE = 20;
    private final int BOARD_WIDTH = 10;
    private final int BOARD_HEIGHT = 22;
```

```
    private final BufferedImage img;
    private Tetrominoe[] board;

    private int curX;
    private int curY;
    private Block curBlock;

    private long score;
    private boolean stopped;

    public Tetris() {
        stopped = true;
        board = new Tetrominoe[BOARD_WIDTH * BOARD_HEIGHT];
        for (int i = 0; i < BOARD_HEIGHT * BOARD_WIDTH; i++) {
            board[i] = Tetrominoe.X;
        }
        curBlock = Block.empty;
        img = new BufferedImage(
                UNIT_SIZE * BOARD_WIDTH,
                UNIT_SIZE * BOARD_HEIGHT,
                BufferedImage.TYPE_INT_BGR);
        draw();
    }
```

今回はWebSocketによりサーバ側でゲーム画像を生成してクライアントで表示する形とするため、`BufferedImage img` にゲームのフレーム画像を書き込みます。

ゲーム盤面は、`Tetrominoe[] board` という配列で保持します。

10 x 22 マスの単一ブロックのゲーム盤面になります。 コンストラクタ中で、空を意味する `Tetrominoe.X` で初期化しています。

```
        board = new Tetrominoe[BOARD_WIDTH * BOARD_HEIGHT];
        for (int i = 0; i < BOARD_HEIGHT * BOARD_WIDTH; i++) {
            board[i] = Tetrominoe.X;
        }
```

ゲームのロジック部分は以下のようになります。

```
public class Tetris {

    // ...

    public void start() {
        for (int i = 0; i < BOARD_HEIGHT * BOARD_WIDTH; i++) {
            board[i] = Tetrominoe.X;
        }
        score = 0;
        stopped = false;
        newPiece();
    }

    public void tick() {
        update();
        draw();
    }

    public void keyPressed(int keycode) {
        if (curBlock.getType() == Tetrominoe.X || stopped) {
            return;
        }
        switch (keycode) {
            case 37 -> tryMove(curBlock, curX - 1, curY); // left arrow
            case 39 -> tryMove(curBlock, curX + 1, curY); // right arrow
            case 40 -> tryMove(curBlock.rotateRight(), curX, curY); // up arrow
            case 38 -> tryMove(curBlock.rotateLeft(), curX, curY); // down arrow
            case 32 -> dropDown(); // Space
        }
    }

    public boolean isStarted() {
        return !stopped;
    }

    private void dropDown() {
        int newY = curY;
```

```
        while (newY > 0) {
            if (!tryMove(curBlock, curX, newY - 1)) {
                break;
            }
            newY--;
        }
        blockDropped();
    }

    private void oneLineDown() {
        if (!tryMove(curBlock, curX, curY - 1)) {
            blockDropped();
        }
    }

    private void blockDropped() {
        for (int i = 0; i < 4; i++) {
            int x = curX + curBlock.x(i);
            int y = curY - curBlock.y(i);
            board[(y * BOARD_WIDTH) + x] = curBlock.getType();
        }
        removeCompleteLines();
        if (curBlock == Block.empty) {
            newPiece();
        }
    }

    private void newPiece() {
        curBlock = Block.randomOf();
        curX = BOARD_WIDTH / 2 + 1;
        curY = BOARD_HEIGHT - 1 + curBlock.minY();
        if (!tryMove(curBlock, curX, curY)) {
            curBlock = Block.empty;
            stopped = true;
        }
    }

    private boolean tryMove(Block newBlock, int newX, int newY) {
        for (int i = 0; i < 4; i++) {
```

```
            int x = newX + newBlock.x(i);
            int y = newY - newBlock.y(i);
            if (x < 0 || x >= BOARD_WIDTH || y < 0 || y >= BOARD_HEIGHT) {
                return false;
            }
            if (shapeAt(x, y) != Tetrominoe.X) {
                return false;
            }
        }

        curBlock = newBlock;
        curX = newX;
        curY = newY;

        draw();
        return true;
    }

    private void removeCompleteLines() {

        int completeLineCount = 0;

        for (int i = BOARD_HEIGHT - 1; i >= 0; i--) {
            boolean complete = true;
            for (int j = 0; j < BOARD_WIDTH; j++) {
                if (shapeAt(j, i) == Tetrominoe.X) {
                    complete = false;
                    break;
                }
            }

            if (complete) {
                completeLineCount++;
                for (int k = i; k < BOARD_HEIGHT - 1; k++) {
                    for (int j = 0; j < BOARD_WIDTH; j++) {
                        board[(k * BOARD_WIDTH) + j] = shapeAt(j, k + 1);
                    }
                }
            }
```

```
        }

        if (completeLineCount > 0) {
            score += completeLineCount * completeLineCount * 100L;
        }
        curBlock = Block.empty;
    }


    private void update() {
        if (curBlock == Block.empty) {
            newPiece();
        } else {
            oneLineDown();
        }
    }

    private Tetrominoe shapeAt(int x, int y) {
        return board[(y * BOARD_WIDTH) + x];
    }

}
```

`keyPressed()` により操作に応じたブロックの回転や移動お行い、外部から呼ばれる `tick()` ブロックの落下とゲームの描画が行われます。

ゲームの描画は以下のようになります。

```
public class Tetris {

    public void tick() {
        update();
        draw();
    }

    private void draw() {
        Graphics g = img.getGraphics();
        g.setColor(Color.BLACK);
```

```
        g.fillRect(0, 0, img.getWidth(), img.getHeight());

        for (int i = 0; i < BOARD_HEIGHT; i++) {
            for (int j = 0; j < BOARD_WIDTH; j++) {
                Tetrominoe shape = shapeAt(j, BOARD_HEIGHT - i - 1);
                if (shape != Tetrominoe.X) {
                    drawSquare(g, j * UNIT_SIZE,
                            i * UNIT_SIZE, shape);
                }
            }
        }
        if (curBlock.getType() != Tetrominoe.X) {
            for (int i = 0; i < 4; i++) {
                int x = curX + curBlock.x(i);
                int y = curY - curBlock.y(i);
                drawSquare(g, x * UNIT_SIZE,
                        (BOARD_HEIGHT - y - 1) * UNIT_SIZE,
                        curBlock.getType());
            }
        }
        g.setColor(Color.WHITE);
        g.drawString("Score:" + score, 10, 15);
        g.dispose();
    }

    private void drawSquare(Graphics g, int x, int y, Tetrominoe shape) {

        g.setColor(shape.getColor());
        g.fillRect(x + 1, y + 1, UNIT_SIZE - 2, UNIT_SIZE - 2);

        g.setColor(shape.getColor().brighter());
        g.drawLine(x, y + UNIT_SIZE - 1, x, y);
        g.drawLine(x, y, x + UNIT_SIZE - 1, y);

        g.setColor(shape.getColor().darker());
        g.drawLine(x + 1, y + UNIT_SIZE - 1,
                x + UNIT_SIZE - 1, y + UNIT_SIZE - 1);
        g.drawLine(x + UNIT_SIZE - 1, y + UNIT_SIZE - 1,
                x + UNIT_SIZE - 1, y + 1);
```

```
    }
}
```

`BufferedImage` に対して、現在のボード盤面のブロック、現在操作中のブロックの順で、ブロックの色に応じた四角を描画しているだけです。

`BufferedImage` の内容は、以下でPNG画像として書き込みを行います。

```
    public void write(OutputStream os) throws IOException {
        ImageIO.write(img, "png", os);
        os.flush();
    }
```

WebSocket エンドポイント

`/tetris` というパスでエンドポイントを作成します。

```
@ServerEndpoint("/tetris")
public class TetrisEndPoint {

    private static final Map<Session, Tetris> sessions = new ConcurrentHashMap<>();
    private static final ScheduledExecutorService executor = Executors.newSingleThreadSche
    static {
        Runnable command = () -> sessions.entrySet().forEach(s -> {
            final Tetris tetris = s.getValue();
            synchronized (tetris) {
                if (!tetris.isStarted()) {
                    return;
                }
                tetris.tick();
                TetrisEndPoint.send(s.getKey(), tetris);
            }
        });
        executor.scheduleWithFixedDelay(command, 0, 500, TimeUnit.MILLISECONDS);
```

```
    }

    @OnOpen
    public void onOpen(Session session) {
        final Tetris tetris = new Tetris();
        sessions.put(session, tetris);
        send(session, tetris);
    }

    @OnClose
    public void onClose(Session session) {
        sessions.remove(session);
    }

    @OnMessage
    public void onMessage(String message, Session session) {
        Tetris tetris = sessions.get(session);
        synchronized (tetris) {
            if (!tetris.isStarted()) {
                if (message.equals("83")) { // s:start
                    tetris.start();
                }
            } else {
                tetris.keyPressed(Integer.parseInt(message));
            }
            send(session, tetris);
        }
    }

    private static void send(Session session, Tetris tetris) {
        try {
            OutputStream os = session.getBasicRemote().getSendStream();
            tetris.write(os);
            os.close();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

```
}
```

`ScheduledExecutorService` で定期的に `tick()` でゲームを進める。 `onMessage()` でクライアントから送信されたキー操作を扱う。という内容で、前回簡単な例で見たのと同じ形です。

WebSocket クライアント

クライアント側も前回と同様で、WebSocket で取得した画像を反映するだけの形です。

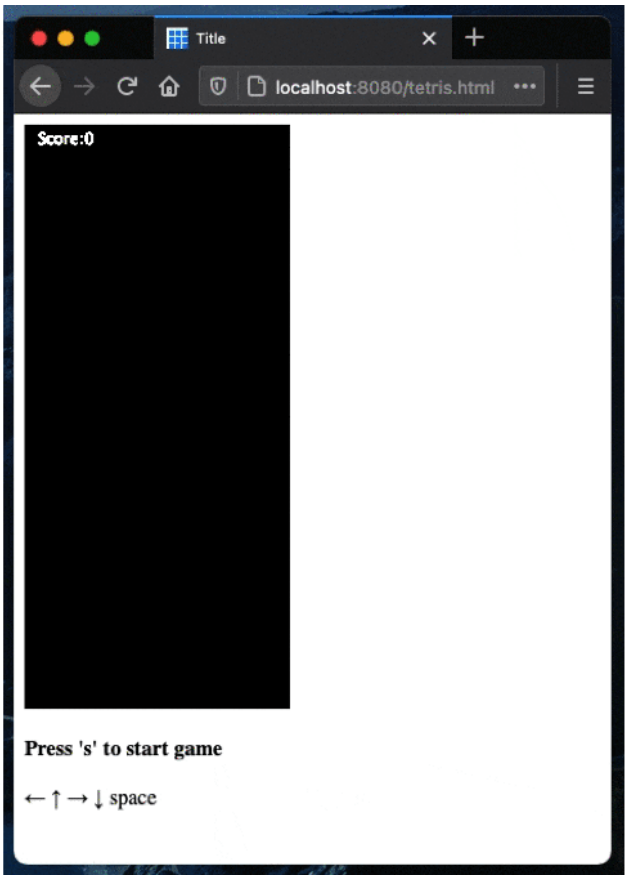
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<img id="img">
<br>
<p><b>Press 's' to start game</b></p>
<p> ← ↑ → ↓ space</p>

<script>
const socket = new WebSocket('ws://localhost:8025/websockets/tetris');
socket.addEventListener('message', (event) => {
    const url = window.URL || window.webkitURL;
    document.getElementById('img').src = url.createObjectURL(event.data);
});

document.addEventListener('keydown', (event) => {
    socket.send(event.keyCode);
});
</script>
```

```
</body>
</html>
```

ゲームの実行

ゲームを実行してみましょう。

`s` キーでスタートします。



よさそうですね。

まとめ

3回に渡り、WebSocket の使い方と、サーバサイドで(画像を)レンダリングするゲームの実装について見てきました。

今回の例では Javascript にてクライアント側でクローズする話ではあるのですが、あえてサーバサイドで生成した画像でテトリスを作ってみました。