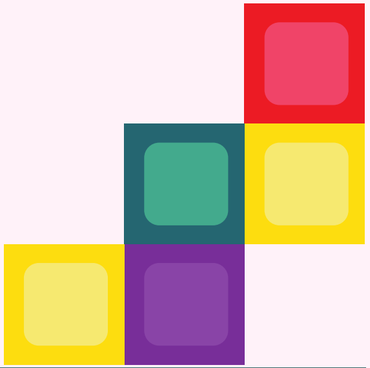
OLD SCHOOL TETRIS

DELA CRUZ, PERILLO, RODRIGUEZ

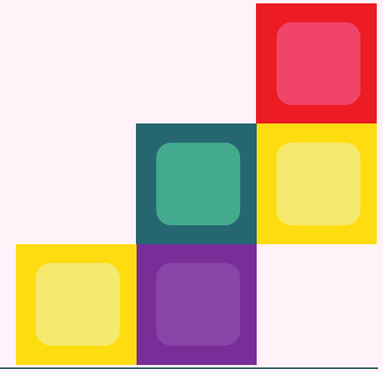
RECAP

- *The team set out to celebrate the anniversary of Tetris with a throwback to the first few iterations of Tetris with what modern spins the team could add.*
- *The main goal was to apply OS concepts and produce a recreation of the game.*
- *What follows is the product of the team's work.*



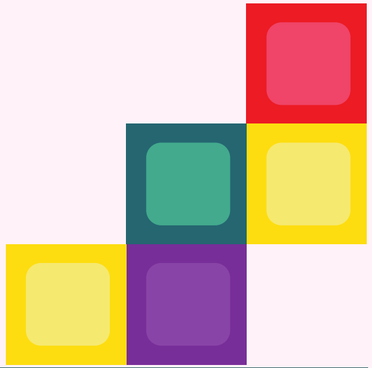
DISCLAIMER

- *Unfortunately, due to limitations the group was not able to implement some features initially presented.*
- *Local Co-Op: This feature is not implemented due to time restrictions.*



FIRST LOOK

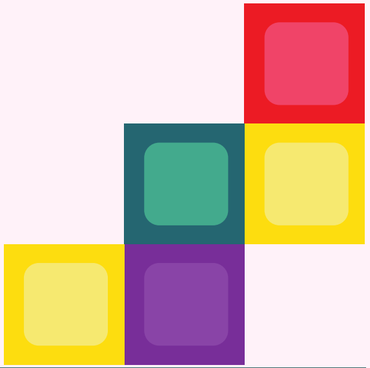
- *The game can be started by clicking on the Tetris.exe file within the master folder.*
- *Wait for the jump scare to boot up and then a separate window loads the screen of the game along with the instructions on how to play, the key binds, the high score and current score.*



CONCEPTS APPLIED (GAME.JAVA)

- *Thread Implementation which is present via the Game.java file in the folder*

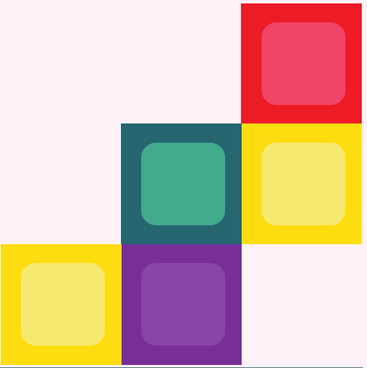
```
//Gameplay Threads
private Thread threadAnimation;
private Thread threadAutoDown;
private Thread threadLoaded;
private long playTime;
private long lTimeStep;
final static int INPUT_DELAY = 40;
private boolean bMuted = true;
private boolean isRestarting = false;
```



CONCEPTS APPLIED (GAME.JAVA)

- *Concurrency Control via implementing the restartGame() method that prevents from restarting too soon*

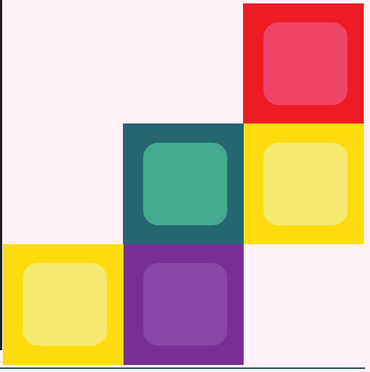
```java
public void restartGame() {
    long currentTime = System.currentTimeMillis();
    if (currentTime - lastRestartTime < RESTART_COOLDOWN || isRestarting) {
        return; // Ignore restart if too soon or already restarting
    }

    isRestarting = true;
    lastRestartTime = currentTime;

    try {
        // Stop all existing threads first
        stopThreads();

        // Reset the game state
        GameLogic.getInstance().clearBoard();
        GameLogic.getInstance().initGame();
        GameLogic.getInstance().setbPlaying(bPlaying:true);
        GameLogic.getInstance().setbPaused(bPaused:false);
        GameLogic.getInstance().setbGameOver(bGameOver:false);
        GameLogic.getInstance().setbRestarted(bRestarted:true);

        // Reset the screen
        gmsScreen.resetGame();
```



CONCEPTS APPLIED (GAME.JAVA)

- *The run() method sets the thread priority to Thread.MIN_PRIORITY to manage CPU allocation among threads.*

```java
public void run(){
    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);

    long lStartTime = System.currentTimeMillis();
    if(!GameLogic.getInstance().isbLoaded() && Thread.currentThread() == threadLoaded){
        GameLogic.getInstance().setbLoaded(bLoaded:true);
    }

    while(Thread.currentThread() == threadAutoDown){
        if(!GameLogic.getInstance().isbPaused() && GameLogic.getInstance().isbPlaying()){
            tryMovingDown();
        }
        gmsScreen.repaint();
        try{
            lStartTime += nAutoDelay;
            Thread.sleep(Math.max(a:0, lStartTime - System.currentTimeMillis()));
        } catch (InterruptedException e){
            break;
        }
    }
```



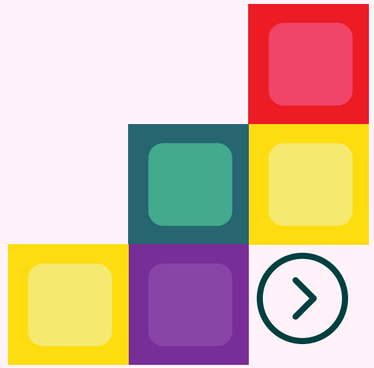
CONCEPTS APPLIED (GAME.JAVA)

The code uses flags such as isRestarting and bMuted to manage the state of the game and ensure that operations like restarting the game or playing music are performed correctly and without conflict

```java
public void restartGame() {
    long currentTime = System.currentTimeMillis();
    if (currentTime - lastRestartTime < RESTART_COOLDOWN || isRestarting) {
        return; // Ignore restart if too soon or already restarting
    }

    isRestarting = true;
    lastRestartTime = currentTime;
```

```java
// Reset music if not muted
if(!bMuted){
    if(clipBGM.isRunning()) {
        clipBGM.stop();
    }
    clipBGM.setFramePosition(frames:0);
    clipBGM.loop(Clip.LOOP_CONTINUOUSLY);
}
```

Team Robby D



GAME.JAVA

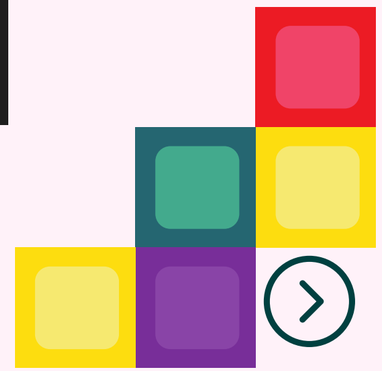
- *Several OS concepts were applied in this main file, especially since this file majorly handles all that happens in the game.*
- *The class contains multiple threads that are responsible in different aspects of the game.*

CONCEPTS APPLIED (GAMESCREEN.JAVA)

The class extends Panel is responsible for the game elements, which includes the grid, tetrominoes, and the score.

```
public class GameScreen extends Panel {
    private Dimension dimOff;
    private Image imgOff;
    private Graphics grpOff;
    public Grid grid = new Grid();
    private GameFrame gmf;
    private Font font = new Font(name:"Monospaced", Font.PLAIN, size:12);
    private Font fontBig = new Font(name:"Monospaced", Font.PLAIN + Font.ITALIC, size:36);
    private final Color retroGreen = new Color(rgb:0xd3f0cb);
    private final Color borderGreen = new Color(rgb:0x2E8B57); // Slightly darker green for border
    private final Color customBackground = new Color(rgb:0x55552e);
    private FontMetrics fontMetrics;
    private int nFontWidth;
    private int nFontHeight;
    private String strDisplay = "";
    public Tetronimo tetronimoOnDeck;
    public Tetronimo tetronimoCurrent;
    private Timer timer;
```

Team Robby D

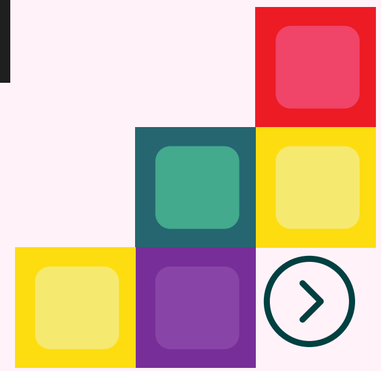


CONCEPTS APPLIED (GAMESCREEN.JAVA)

Includes a KeyAdapter to listen for key events, such as pressing 'R' to restart the game. This is an example of handling asynchronous user input events, which is a common concept in GUI applications.

```
this.addKeyListener(new KeyAdapter() {
    @Override
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_R) {
            restartGame();
        }
    }
});
```

Team Robby D

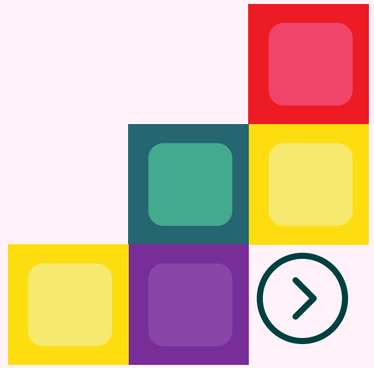


CONCEPTS APPLIED (GAMESCREEN.JAVA)

A Timer is used to manage periodic actions, which could be related to game updates or animations. While not explicitly shown in the viewed lines, timers are often used in games to handle regular updates without blocking the main thread.

```
public class GameScreen extends Panel {
    private Dimension dimOff;
    private Image imgOff;
    private Graphics grpOff;
    public Grid grid = new Grid();
    private GameFrame gmf;
    private Font font = new Font(name:"Monospaced", Font.PLAIN, size:12);
    private Font fontBig = new Font(name:"Monospaced", Font.PLAIN + Font.ITALIC, size:36);
    private final Color retroGreen = new Color(rgb:0xd3f0cb);
    private final Color borderGreen = new Color(rgb:0x2E8B57); // Slightly darker green for border
    private final Color customBackground = new Color(rgb:0x55552e);
    private FontMetrics fontMetrics;
    private int nFontWidth;
    private int nFontHeight;
    private String strDisplay = "";
    public Tetronimo tetronimoOnDeck;
    public Tetronimo tetronimoCurrent;
    private Timer timer;
```

Team Robby D



GAMESCREEN.JAVA

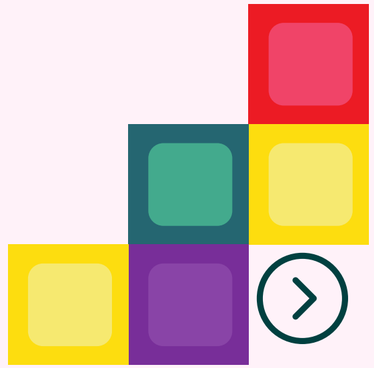
- *Although the file doesn't directly implement threading, it interacts with the game state managed by the Game.java file, which does use threads. The update method logs the game state and updates the screen based on the game's current status.*
- *This class allows user to interact with the game and have feedbacks for their inputs.*

CONCEPTS APPLIED (GAMELOGIC.JAVA)

The class uses a singleton pattern to ensure that only one instance of GameLogic exists. This is achieved through a private constructor and a static getInstance() method. This design pattern is often used to manage shared resources or state in a thread-safe manner, although this implementation does not explicitly handle thread safety.

```
public static GameLogic getInstance() {
    if (instance == null){
        instance = new GameLogic();
    }
    return instance;
}
```

Team Robby D

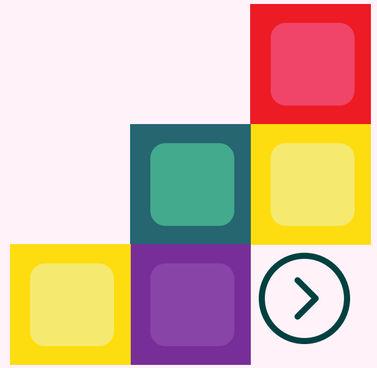


CONCEPTS APPLIED (GAMELOGIC.JAVA)

Methods like checkbThreshold() adjust the game's difficulty based on the score, which indirectly affects the game's threading logic by altering Game.nAutoDelay, a variable that influences thread sleep times for game updates.

```
public void checkbThreshold(){
    if(bScore > bThreshold && Game.nAutoDelay > 30){
        bThreshold += Game.THRESHOLD;
        Game.nAutoDelay -= 15;
    }
}
```

Team Robby D

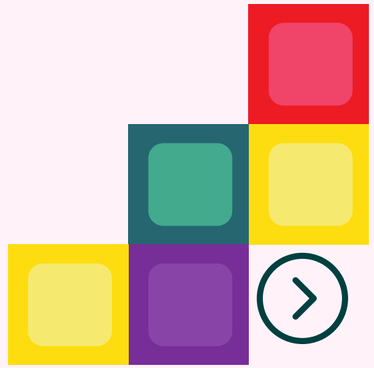


CONCEPTS APPLIED (GAMELOGIC.JAVA)

The class maintains various flags (bPlaying, bPaused, bLoaded, bGameOver, bRestarted) to track the state of the game. These flags are crucial for controlling the flow of the game and interacting with threads that manage game updates.

```java
public boolean isbPlaying() {
    return bPlaying;
}

public void setbPlaying(boolean bPlaying) {
    this.bPlaying = bPlaying;
}

public boolean isbPaused() {
    return bPaused;
}

public void setbPaused(boolean bPaused) {
    this.bPaused = bPaused;
}

public boolean isbRestarted() {
    return bRestarted;
}

public void setbRestarted(boolean bRestarted) {
    this.bRestarted = bRestarted;
}

public boolean isbLoaded() {
    return bLoaded;
}
```

Team Robby D



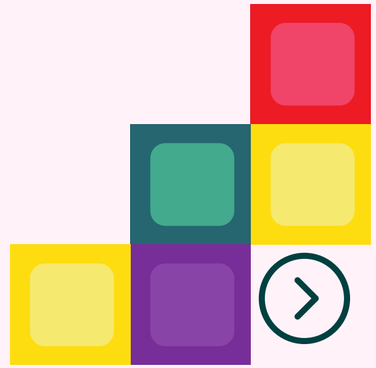
CONCEPTS APPLIED (GAMELOGIC.JAVA)

The resetGame() and initGame() methods reset and initialize the game state, ensuring that the game starts with a clean slate. This is important for ensuring that threads operate on a consistent and predictable game state.

```java
// Reset game state
public void resetGame() {
    bScore = 0;
    bGameOver = false;
    bPlaying = true;
    bPaused = false;
    bRestarted = true;
```

```java
private GameLogic(){}
public void initGame(){
    setbScore(bScore:0);
    setbThreshold(bThreshold:2400);
}
```

Team Robby D



RECAP: WHAT'S THE POINT?

MULTI-THREADING

- Particularly in game logic, **multi-threading** is a prominent concept not only in our application, but in many modern games to allow for a smooth and seamless experience for the players.
-

CONCURRENCY

- This concept will be applied in the game through ensuring that the multiple active threads are safely able to access resources w/o conflict.
-

I/O OPERATIONS

- This concept deals with the proper handling and management of Input from the user and Output from the game.
-

GAME DEMO

SOURCES

- https://docs.google.com/document/d/1xjcCVQIphVyIDav2YelTkTl1oYnMxZU2-Wwm7SBZr80/edit?usp=sharing {GAME DESIGN DOCUMENT}
- https://www.reddit.com/r/gamedev/comments/44fux4/multi_threading_in_game_development/?rdt=49271
- https://github.com/jahnagoldman/tetris-game
- https://docs.oracle.com/javase/tutorial/uiswing/components/splitpane.html
- https://www.reddit.com/r/java/comments/xztgwj/discussion_which_ide_is_best_for_java/
  - https://youtu.be/025QFeZfeyM?si=JG46hjCIyEkVgMIW
- https://develop.games
- https://docs.google.com/document/d/1Vl7BMvzUOhbunJrI_X1gUc6x-LAp3aaBiPwHUf27B70/edit?usp=sharing