assignment # group-project

Sprint 1 – GUI Foundation

School of Engineering and Technology, University of Washington Tacoma
TCSS 305 Programming Practicum, Winter 2026
Value: 10% of the course grade

Due Date

Sunday, 1 March 2026, 23:59:59

Description

Sprint 1 transitions your team from setup to implementation. You will build the foundational GUI for your Tetris game: a visible window with a file menu, a game board panel, a next piece panel, and an information panel. By the end of this sprint, your GUI should display all seven Tetris shapes and frozen blocks on the board, the next piece in its own panel, and provide basic menu functionality.

The game will not be playable yet — Sprint 2 adds game logic and controls. This sprint focuses on layout, drawing, and establishing the visual foundation.

Learning Objectives

By completing this sprint, you will:

- Build a Swing GUI application with menus, dialogs, and multiple panels
- Use layout managers to organize a multi-region interface
- Render game elements using Java 2D graphics (`paintComponent`, `Graphics2D`)
- Read and apply a provided API to draw Tetris pieces and frozen blocks
- Continue practicing collaborative Git workflows (branching, merging, code review)

Before You Begin

Ensure you have completed:

- [x] All Sprint 0 requirements (repository setup, API familiarization, console UI)
- [x] Merged `sprint0` into `main`
- [x] Read the Back-End API Documentation
- [x] Reviewed the Swing guides listed below

Requirements

Requirement 1: Branch Setup

Create a branch from your `main` branch called `sprint1`. All Sprint 1 work should be committed to this branch.

> **Caution**
>
> Do NOT commit, merge, or alter the `sprint1` branch after the due date/time. I will ignore any commits after the deadline.

Requirement 2: GUI Window with File Menu

> **Guide**
>
> - *NEW* Building Menus with JMenuBar — JMenuBar, JMenu, JMenuItem, Exit pattern, and JOptionPane
> - Swing API Basics — JFrame setup, window properties, and Swing fundamentals
> - Adding Event Handlers — How to respond to user actions like menu clicks

Create a visible GUI window (`JFrame`) with a **File** menu containing:

Menu Item	Action
New Game	Open a `JOptionPane` stating a new game is starting

Menu Item	Action
Exit	Close the window
About	Open a `JOptionPane` with group member names and other information

Swing Tutorials

- JFrame Tutorial
- Menu Tutorial
- Dialog Tutorial

Requirement 3: Main Content Area

Guide

Swing Layout Managers — Organizing components with BorderLayout, GridLayout, and other managers

The main content area should be separated into different regions. Meet as a group to brainstorm the layout.

Your layout must include the three regions described in Requirements 4–6. Consider how the regions relate to each other in size and position.

Swing Panel Tutorial

JPanel Tutorial

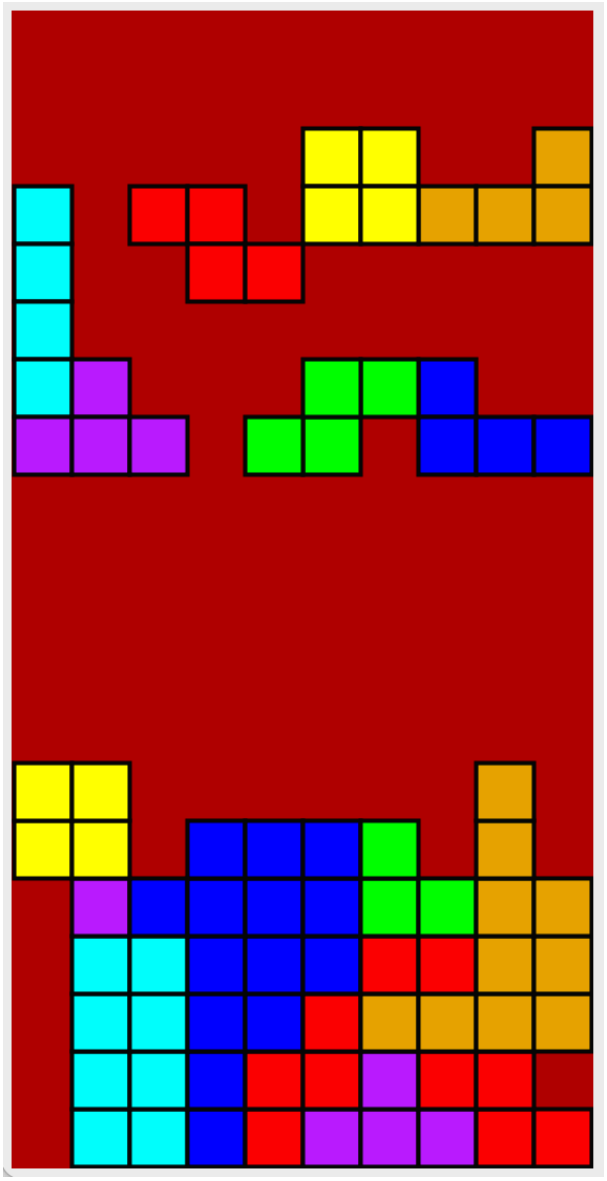
Requirement 4: Tetris Board Region

Guide

NEW Custom Painting with Java 2D — paintComponent, Graphics2D, coordinate system, shapes, and mapping grid coordinates to pixels

- **Background: RED**
- **Dimensions:** 10 wide x 20 high (matching the game model)
- Must draw all 7 Tetris shapes correctly sized and proportionate
  - Example: if the region is 200 x 400 pixels, each block is 20 x 20 pixels
  - Use the provided method that returns an array of `IndividualPiece` objects
- Must draw frozen blocks correctly sized and proportionate
  - Use the provided method that returns a `FrozenBlocks` object

The following image shows an example of what the board region should look like. All 7 Tetris piece types are drawn as movable pieces, and frozen blocks fill the bottom rows. Each block is correctly sized and proportionate to the board dimensions.



To produce this exact output for testing, use the following helper methods to create hardcoded test data. These methods return the same types as the back-end API, so your drawing code should work with both the hardcoded data and the live game model.

pieces()

```
private IndividualPiece[] pieces() {
    return new IndividualPiece[]{
        new IndividualPiece(new Point[]{new Point(9, 17), new Point(7, 16),
                new Point(8, 16), new Point(9, 16)}, Block.L),
        new IndividualPiece(new Point[]{new Point(5, 17), new Point(6, 17),
                new Point(5, 16), new Point(6, 16)}, Block.O),
        new IndividualPiece(new Point[]{new Point(2, 16), new Point(3, 16),
                new Point(3, 15), new Point(4, 15)}, Block.Z),
        new IndividualPiece(new Point[]{new Point(0, 16), new Point(0, 15),
                new Point(0, 14), new Point(0, 13)}, Block.I),
        new IndividualPiece(new Point[]{new Point(7, 13), new Point(7, 12),
                new Point(8, 12), new Point(9, 12)}, Block.J),
```

```
        new IndividualPiece(new Point[]{new Point(1, 13), new Point(0, 12),
                new Point(1, 12), new Point(2, 12)}, Block.T),
        new IndividualPiece(new Point[]{new Point(5, 12), new Point(4, 12),
                new Point(6, 13), new Point(5, 13)}, Block.S)
    };
}
```

frozenBlocks()

```
private GameControls.FrozenBlocks frozenBlocks() {
    return new GameControls.FrozenBlocks(List.of(
        new Block[]{null, Block.I, Block.I, Block.J, Block.Z, Block.T,
                Block.T, Block.T, Block.Z, Block.Z},
        new Block[]{null, Block.I, Block.I, Block.J, Block.Z, Block.Z,
                Block.T, Block.Z, Block.Z, null},
        new Block[]{null, Block.I, Block.I, Block.J, Block.J, Block.Z,
                Block.L, Block.L, Block.L, Block.L},
        new Block[]{null, Block.I, Block.I, Block.J, Block.J, Block.J,
                Block.Z, Block.Z, Block.L, Block.L},
        new Block[]{null, Block.T, Block.J, Block.J, Block.J, Block.J,
                Block.S, Block.S, Block.L, Block.L},
        new Block[]{Block.O, Block.O, null, Block.J, Block.J, Block.J,
                Block.S, null, Block.L, null},
        new Block[]{Block.O, Block.O, null, null, null, null, null, null,
                Block.L, null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null},
        new Block[]{null, null, null, null, null, null, null, null, null,
null}
    ));
}
```

API Reference

Back-End API Documentation

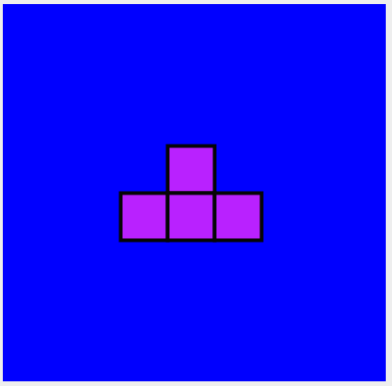
Requirement 5: Next Piece Region

Guide

NEW Custom Painting with Java 2D — paintComponent, Graphics2D, coordinate system, shapes, and mapping grid coordinates to pixels

- **Background: BLUE**
- **Shape:** Square, smaller than the board
- Must draw the T tetromino centered inside the region
- Use the provided method that returns an `IndividualPiece` object for the next piece

The following image shows an example of what the next piece region should look like. The T piece is centered within the blue square panel.



nextPiece()

```
private IndividualPiece nextPiece() {
    return new IndividualPiece(
        new Point[]{new Point(1, 2), new Point(0, 1),
                new Point(1, 1), new Point(2, 1)},
        Block.T
    );
}
```

> **Note**
>
> For this sprint, you only need to display the T piece in the next piece panel. In Sprint 2, you will connect this panel to the model via `PropertyChangeListener` so it updates dynamically.

Requirement 6: Information Region

- **Background: GREEN**
- Smaller than the board
- May be empty for this sprint (will contain score, controls, key bindings, etc. in later sprints)

Requirement 7: Executive Summary

Your project includes a file called `executive-summary.md`. This file documents your team's contributions, meeting notes, and commentary for each sprint.

> **External Resource**
>
> Markdown Cheat Sheet

Add the following sections for Sprint 1:

- **Sprint 1 Contributions** — for each group member, describe their contribution to Sprint 1
- **Sprint 1 Comments** — document any issues, ideas, code weirdness, etc.

> **Important**
>
> Keep all previous sprint entries intact. The executive summary is a living document.

🕮 Guide Reference

Guide	Description
NEW Building Menus with JMenuBar	JMenuBar, JMenu, JMenuItem, Exit pattern, and JOptionPane
NEW Custom Painting with Java 2D	paintComponent, Graphics2D, coordinate system, shapes, and mapping grid coordinates to pixels
Swing API Basics	JFrame setup, window properties, and Swing fundamentals
Swing Layout Managers	Organizing components with BorderLayout, GridLayout, and other managers
Adding Event Handlers	How to respond to user actions like menu clicks
Git Version Control	Git fundamentals: commits, branches, and remote repositories
Git Branching and Pull Requests	How to work with branches, merge changes, and submit via pull request

---

🚀 Submission and Grading

> **Important**
>
> Start early — GUI layout and drawing require iteration and testing across team members.

Please see the rubric in Canvas for a breakdown of the grade for this sprint.

Submitting Your Work

1. Ensure all Sprint 1 requirements are committed and pushed to your `sprint1` branch
2. Perform one merge from `sprint1` into your `main` branch
3. Push the merge to the remote repository
4. Submit the GitHub repository link to the assignment in Canvas

I will look at both the `sprint1` and `main` branches. I will ignore any commits after the due date.

Tip

You can commit and push multiple times. Only your final push before the deadline will be graded.