assignment # group-project

Sprint 2 – Game Logic & Controls

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

Due Date

Sunday, 8 March 2026, 23:59:59

Description

Sprint 2 brings your Tetris GUI to life. You will connect the visual components from Sprint 1 to the back-end model using the **Observer Design Pattern** (`PropertyChangeListener`), add keyboard controls so the player can move and rotate pieces, and drive the game forward with a `javax.swing.Timer` .

The key architectural principle in this sprint: **the GUI never manipulates game state directly.** Every user action calls a single method on the `GameControls` API, and every visual update is triggered by a `PropertyChangeEvent` from the model. This clean separation between front-end and back-end is the Observer pattern in action.

Learning Objectives

By completing this sprint, you will:

- Apply the Observer Design Pattern using `PropertyChangeListener` and `PropertyChangeEvent`
- Connect GUI components to a back-end model through event-based communication
- Use `javax.swing.Timer` to drive periodic game updates
- Implement keyboard input handling with `KeyListener`
- Practice the principle of separation of concerns between front-end and back-en

Before You Begin

Ensure you have completed:

- [x] All Sprint 1 requirements (GUI window, board panel, next piece panel, information panel)
- [x] Merged `sprint1` into `main`
- [x] Reviewed the Back-End API Documentation
- [x] Read about GameControls and GameEvent in the API docs

Requirements

Requirement 1: Branch Setup

Create a branch from your `main` branch called `sprint2`. All Sprint 2 work should be committed to this branch.

> **Caution**
>
> Do NOT commit, merge, or alter the `sprint2` branch after the due date/time. I will ignore any commits after the deadline.

Requirement 2: Top-Level GUI Class Setup

> **Guide**
>
> - *NEW* Animation with javax.swing.Timer — EDT-safe timers, start/stop, and the game loop
> - *NEW* The Observer Pattern — PropertyChangeListener, PropertyChangeSupport, and event-driven updates
> - *NEW* Model-View-Controller (MVC) — Understanding the architecture: Model (TetrisGame), View (panels), Controller (key/menu handlers)

In the top-most GUI class (`JFrame` or top-level `JPanel`):

Model Reference:

- Add a field that holds a reference to the primary model object (`TetrisGame`)

- **Only instantiate one `TetrisGame` object** — this single instance is shared across all components that need it

Timer Setup:

- Instantiate a `javax.swing.Timer` and store a reference to it
- Determine an appropriate tick interval (how often the timer fires)
- Add an `ActionListener` to the Timer that calls the appropriate method in the `GameControls` API to advance the game by one step

> ⚠ **Single Responsibility**
>
> The Timer's `ActionListener` should **ONLY** call a single method in the `GameControls` API. It should NOT call `repaint()`, move shapes, update scores, or do any other work. The model will fire `PropertyChangeEvent`s that trigger those updates.

Requirement 3: PropertyChangeListener for GameState

> 📖 **Guide**
>
> - *NEW* The Observer Pattern — How PropertyChangeListener connects the model to the GUI
> - *NEW* Animation with javax.swing.Timer — Starting and stopping the timer in response to GameState changes

Implement `PropertyChangeListener` in the top-level GUI class and register it with the `TetrisGame` object. Listen for game state changes:

- `GameEvent.GameStateChanged`
- `GameControls.GameState`

Respond to each state:

GameState	Action
`NEW`	Set up the GUI for a new game (this may be nothing to do)
`RUNNING`	Start the Timer

GameState	Action
`PAUSED`	Stop the Timer
`OVER`	No requirement for this sprint

> **Tip**
>
> Think about what "set up the GUI for a new game" means. If the user starts a second game, what needs to reset?

Requirement 4: Menu Integration

Connect the **New Game** menu item from Sprint 1 to the back-end. When the user selects New Game, call the correct method from the `GameControls` API.

> **Single Responsibility**
>
> The menu action should **ONLY** call a single method in the `GameControls` API — do not directly start timers, reset panels, or change any GUI state. The model's `PropertyChangeEvent`s will handle those updates through the listeners you implement in other requirements.

Requirement 5: KeyListener for Game Controls

> **Guide**
>
> - *NEW* Handling Key Events — KeyListener, KeyAdapter, and three patterns for mapping keys to actions (if-chain, Map, switch expression)

Add a `KeyListener` to the top-most GUI `JPanel`. Each key press should **ONLY** call a single method in the `GameControls` API — no direct GUI changes.

Action	Key Handler
Move piece left	Call appropriate `GameControls` method

Action	Key Handler
Move piece right	Call appropriate `GameControls` method
Move piece down one	Call appropriate `GameControls` method
Drop piece	Call appropriate `GameControls` method
Rotate piece	Call appropriate `GameControls` method
Pause game	Call appropriate `GameControls` method

Tip

Pay attention to the reported `GameState` of the `TetrisGame` to know when the game is paused. Your key handler should check the current state — some actions should be ignored when the game is paused.

Requirement 6: Board Panel PropertyChangeListener

Guide

NEW The Observer Pattern — Implementing PropertyChangeListener to receive model updates

In the GUI class containing the Tetris Board visual (`JPanel`):

1. Implement `PropertyChangeListener`
2. Register this object with the `TetrisGame` 's property change support
3. Listen for:
    - `GameEvent.CurrentPieceChanged` — the movable piece has moved or changed
    - `GameEvent.FrozenBlocksChanged` — the frozen blocks have changed
4. Paint/draw/fill the current moving piece and frozen blocks
5. The display must update when new `PropertyChangeEvent` s are received

> **Note**
>
> In Sprint 1, you drew pieces using hardcoded test data. Now, your `paintComponent` code receives live data from the model via property change events. Your drawing logic from Sprint 1 should work with minimal changes — the data types are the same.

Requirement 7: Next Piece Panel PropertyChangeListener

> **Guide**
>
> *NEW* The Observer Pattern — Implementing PropertyChangeListener to receive model updates

In the GUI class containing the Next Piece visual (`JPanel`):

1. Implement `PropertyChangeListener`
2. Register this object with the `TetrisGame` 's property change support
3. Listen for:
   - `GameEvent.NextPieceChanged` — the next piece has changed
4. Paint/draw/fill the next piece — it must be visible, centered, identifiable, and update correctly

---

Not Required for Sprint 2

The following are **not** required for this sprint:

- Ending a game (reacting to `GameState.OVER`)
- Scoring or level tracking
- Finalized styles or look and feel
- Extra features (sound, themes, high scores, etc.)

These will be addressed in Sprint 3.

Requirement 8: Daily Meetings (2 Required)

Hold at least **2** daily meetings during this sprint. Create a Google Doc to document your meetings. Each meeting should contain:

- Date and time
- Members who attended
- Location/method (Discord voice/video, Zoom, in-person)
- Each group member answers:
    a. What did you do since the last meeting?
    b. What will you do before the next meeting?
    c. Is there anything the group can do to help you?

Put the link to the Google Doc in your `README.md` (with view access).

> **Tip**
>
> These meetings mirror **daily stand-ups** used in professional Agile/Scrum teams. Keep them focused — 15 minutes is a good target.

Requirement 9: Executive Summary

Add the following sections to your `executive-summary.md`:

- **Sprint 2 Contributions** — for each group member, describe their contribution to Sprint 2
- **Sprint 2 Meetings** — link to meeting minutes, describe other meetings (where/when/how, what was discussed), describe primary forms of communication besides formal meetings
- **Sprint 2 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 Introduction to Design Patterns	What design patterns are, why they matter, and the patterns used in this project
NEW The Observer Pattern	PropertyChangeListener, PropertyChangeSupport, firing and handling property change events
NEW Model-View-Controller (MVC)	Separating Model (game logic), View (panels), and Controller (input handling)
NEW The Strategy Pattern	Map-based strategy pattern used in the back-end API for move validation
NEW Handling Key Events	KeyListener, KeyAdapter, and three patterns for mapping keys to game actions
NEW Animation with javax.swing.Timer	EDT-safe timer for driving the game loop, start/stop/setDelay
Adding Event Handlers	ActionListener patterns for menu items and timer callbacks

🚀 Submission and Grading

> **Important**
>
> Start early — connecting the model to the GUI requires coordination across team members. Plan who is implementing which listeners.

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

Submitting Your Work

1. Ensure all Sprint 2 requirements are committed and pushed to your `sprint2` branch
2. Perform one merge from `sprint2` into your `main` branch
3. Push the merge to the remote repository

I will look at both the `sprint2` and `main` branches. I will ignore any commits after the due date.

> **Tip**
>
> You can commit and push to your `sprint2` branch as often as you like. Only perform one merge into `main` at the end.