Patterns

Table of Contents

Patterns

Model-View-Controller

Pattern Name	Model-View-Controller (MVC)
Aliases	N/A
Type	Architectural Pattern
Reference	https://www.geeksforgeeks.org/mvc-design-pattern/
Problem	The user needs to play the game through an intuitive mouse-based GUI.
Motivation	Any situation where the user applies a mouse input on a GUI element in the game to change the internal game state.
Context	Since the target audience is early high schoolers, the user wants to interact with the game menus and puzzles using a mouse, with close to no keyboard input. The user can navigate and play the game using GUI elements such as buttons. The user wants to minimize keyboard inputs.
Forces	The game must have a GUI navigation system implemented. The game has a secondary keyboard navigation system.
Solution	The game imports the JavaX Swing library, which supports buttons, text boxes, and draggable GUI containers. Swing handles the hardware inputs (controller) and provides GUI elements (View). The backend classes (model) do not use Swing and purely handle the game logic and file access.
Intent	This separation of frontend and backend development facilitates frequent improvements in UI / UX design without affecting the core game logic.
Collaborations	N/A
Consequences	1. Developers must spend time learning MVC before implementing it effectively. 2. The MVC implementation adds development overhead due to the extra complexity of the code. 3. The extra communication between components (Model, View, Controller) may decrease the runtime performance of the game.
Implementation	N/A
Known Uses	JavaX Swing
Related Patterns	Hierarchical model–view–controller (HMVC) Model–view–adapter (MVA) Model–view–presenter (MVP) Model–view–viewmodel (MVVM)

Lazy Loading

Pattern Name	Lazy Loading
Aliases	Asynchronous Loading
Type	Architectural Pattern
Reference	https://www.geeksforgeeks.org/lazy-loading-design-pattern/

Problem	The teacher needs to see students' saved progression within 3 seconds of lag.
Motivation	Any time the teacher opens the progression screen to view all students' statistics.
Context	Instead of opening each user data file in a text editor, the teacher wants to have a GUI overview listing all the students' current progression in the game. The teacher can view detailed statistics of a student's progression in each stage by clicking on that entry.
Forces	The teacher wants to view both general and detailed statistics of each student. The game cannot lag for more than 3 seconds.
Solution	The users' data is stored as multiple files, one file per user (student), and the progression page only displays up to 10 entries at a time. When the teacher clicks the "next page" button or updates the search text, the old entries are unloaded and at most 10 other entries are loaded. In this case, only the first few lines of each file are partially loaded, which contain the general statistics of each student. The file is fully loaded when the teacher clicks on an entry to view the detailed, per-stage statistics of that particular student. It is unloaded when the teacher clicks the "back to list" button.
Intent	This partial loading approach reduces lag from fetching non-essential data.
Collaborations	Flyweight: Further reduces lag by reducing the number of possible objects to load.
Consequences	1. Developers must spend time learning Lazy Loading before implementing it effectively. 2. Data access becomes more complicated, as the game must check if the required data is partially or fully loaded.
Implementation	N/A
Known Uses	AngularJS Firefox (via HTML attributes)
Related Patterns	N/A

Flyweight

Pattern Name	Flyweight
Aliases	N/A
Type	Object-Oriented Pattern
Reference	https://www.geeksforgeeks.org/flyweight-design-pattern/
Problem	The user needs to see a graphical representation of the maze up to grid size 100 x 100.
Motivation	Any stage with a large number of maze objects and/ or action blocks.
Context	Instead of seeing a text-based representation of the maze/ action chain, each maze item/ action chain has a unique image icon (sprite) to represent it graphically. The icons move around the screen based on the (updated) position of the maze item/ action chain.
Forces	The game needs to handle up to around 10^5 sprites on screen. The game cannot lag for more than 3 seconds.
Solution	The game stores the sprites (image icons) of the action blocks and maze items in two hashmaps. When creating a new instance of a maze item/ action block, the game first checks the corresponding hashmap before creating a new sprite object.
Intent	This hashmapping approach reduces memory usage by not storing repeated instance objects.
Collaborations	Lazy Loading: Further reduces memory usage by only loading objects that are immediately needed.
Consequences	1. Developers must spend time learning Flyweight before implementing it effectively.
Implementation	N/A
Known Uses	LibreOffice Writer
Related Patterns	GOF Design Patterns