**Software Design Patterns**

**Project: “Minesweeper”**

**Participants: Alsalim, Maxim, Makhambet**

**Contents:**

**1 System architecture overview………………………… p.3**

**2 Design Patterns and their role………………………… p.3**

**3 Usage Instructions…………………………………………… p.5**

**4 System Assumptions and Limitations……………… p.5**

**Goal:** using software patterns implement “minesweeper” game; connect all patterns in one working structure; implement logic and architecture of project.

1. **System architecture overwiev:**

The project is structured using the **MVC (Model-View-Controller)** architectural pattern to separate logic, interface, and game event management. Main components include:

\* **Model:** Manages game logic, including grid layout, cell states, and game rules.

**\* View:** Displays the game board, updates cell status, and provides feedback.

**\* Controller:** Accepts user input, directs requests to the Model, sets timer, and updates the View.

**2 Design Patterns and their role:**

**2.1 Singleton Pattern**

The game logic manager is implemented as a Singleton to ensure a single game instance controls the board, mines placement, and endgame conditions.

**2.2 Factory Method Pattern**

Used to create cells through the **CellFactory** class, where cells vary by type (MineCell, NumberCell, EmptyCell). This pattern provides flexibility and a clean way to create objects with different properties and behaviors.

**2.3 Observer**Updates the interface whenever a cell's state changes. When a cell status (opened or flagged) is updated, View receives notifications from Model and reflects the changes in real time.

**2.4 The Proxy**

Controls access to the cells on the board. Instead of directly interacting with the cell objects, the game uses a CellProxy class that manages access to the actual cell data. This pattern helps to optimize game performance.

**2.5 State**Manages different game states. "Playing," "Victory," and "Defeat" states help control the game’s flow and manage board interactions, preventing field updates at inappropriate times.

**2.6 Command**Implemented for player actions (e.g., opening a cell, setting a flag). Commands simplify the addition of action history, making it possible to add undo functionality for future development.

**3 Usage Instructions:**

**3.1 Starting the Game**

* Open the main class Main in your IDE and run it to start the game.

**3.2 Gameplay Actions**

* Opening a Cell: Left-click a cell to reveal it. If it is empty, neighboring cells may also be revealed. If it contains a mine, the game ends.
* Setting a Flag: Right-click a cell to mark it with a flag, indicating it may contain a mine.

**4 System Assumptions and Limitations**

**Limitations:**

* **Individual approach on writing code**
* **Contribution a lot of patterns in our project**
* **Project does not require complex structure**

**System Assumptions:**

* **Board Dimensions: The board size should be between 5x5 and 30x30 cells.**
* **Number of Mines: Ideally, the number of mines should not exceed 20% of the total cell count.**