## JayHawk Dream Team

# The Greatest Game of Minesweeper
# Software Architecture Document

**Version 1.1**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 17/09/2025 | 1.0 | Compilation of Our Prior Notes | Carlos Mbendera |
| 21/09/2025 | 1.1 | Final Updates | Carlos Mbendera |
| | | | |
| | | | |
| | | | |
| | | | |

# Table of Contents

# Software Architecture Document

## 1. Introduction

This Software Architecture Document (SAD) outlines the architectural framework and high level design for The Greatest Game of Minesweeper, a pygame version of Minesweeper developed as part of our EECS 581 Coursework at the University of Kansas. This blueprint helps ensure that the development aligns with the Software Requirements Specifications (SRS), guiding the team towards successful implementation.

### 1.1 Purpose

This document delivers a comprehensive architectural overview of The Greatest Game of Minesweeper, capturing key decisions and presenting them via various architectural views. It is designed for the project team, to guide implementation, and for stakeholders seeking detailed insights into the software's capabilities.

### 1.2 Scope

The scope of this document encompasses the complete architecture of theThe Greatest Game of Minesweeper, including Graphics Rendering, Mouse Input, Logic, Data Structures for API interactions and a Game Manager that controls the game’s state.

### 1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **API**: Application Programming Interface
- **UI**: User Interface
- **pygame**: A python framework that offers function calls for features often used in game development

### 1.4 References


- EECS 348 Software Engineering Course Syllabus, Professor Hossein Saiedian, University of Kansas, Spring Semester 2024.
- EECS 581Project 1: Minesweeper System Development Specifications, Professor Hossein Saiedian, University of Kansas, Fall Semester 2025.


### 1.5 Overview

This document covers our software design architecture. It will touch on various items such as size, interface, and structure. It covers:

- **Architectural Representation:** Outlines views and models representing the system.
- **Architectural Goals and Constraints:** Details architectural requirements and objectives.
- **Logical View:** Describes system decomposition into packages and classes.
- **Interface Description:** Provides high-level details of key interfaces.
- **Development Description:** Overview of the hours we worked and our development process
- **Quality:** Explains how the architecture supports quality attributes such as extensibility and reliability.

## 2. Architectural Representation

The software architecture for The Greatest Game of Minesweeper system encompasses multiple views that collectively provide a comprehensive understanding of the system's structure and behavior. Each view focuses on specific aspects of the architecture, facilitating communication among stakeholders and guiding the development and evolution of the system.

**1. Overview:**

**Description**: Provides a high-level overview of the system architecture, highlighting its key components and their interactions.

**Model Elements:**

- Main components of the system (e.g., user interface, input handling, data structures).
- High-level communication pathways between components.
- Deployment environment (e.g., computer details, python and pygame version).

**2. Logical View:**

**Description**: Describes the logical organization of the system, focusing on the decomposition of functionality into modules or layers.

**Model Elements:**

- Modules representing major functional areas (e.g., board.py, renderer.py).
- Relationships and dependencies between modules.
- Interfaces exposed by each module for interaction with other parts of the system.

**3. Process View:**

**Description**: Shows the dynamic behavior of the system, illustrating how processes and threads interact to fulfill user input on the Minesweeper board.

**Model Elements:**

- Processes representing user interactions (e.g., clicking on the board, showing mines/flags).
- Communication pathways between processes (e.g., data structures, imported packages).

**4. Development View:**

**Description**: Focuses on the organization of the development effort, including the structure of source code, version control, and development environments.

**Model Elements:**

- Source code repositories and their organization (e.g., repositories for frontend, backend, testing).
- Development environments (e.g., local development setups, GitHub Projects)

## 3. Architectural Goals and Constraints

- **Developmental tools**: We will utilize Visual Studio Code for its versatility, Git and GitHub for version control and PyCharm / VSCode with the Python Extension, ensuring a robust development environment.
- **Design:** Object Oriented Programming - to facilitate clear modularity and extensibility of the software components.
- **Usability**: Develop an intuitive user interface and solid technical backend to enhance gameplay experience.
- **Reuse**: Architect the codebase to promote code reuse and modularity for easier maintenance and scalability.
- **Off-the-shelf product:** Integrate third-party APIs or libraries for features such as content recommendation

- **Design and implementation strategy:** We plan to follow a waterfall development methodology, to ensure phases and requirements are met in a structured progression.

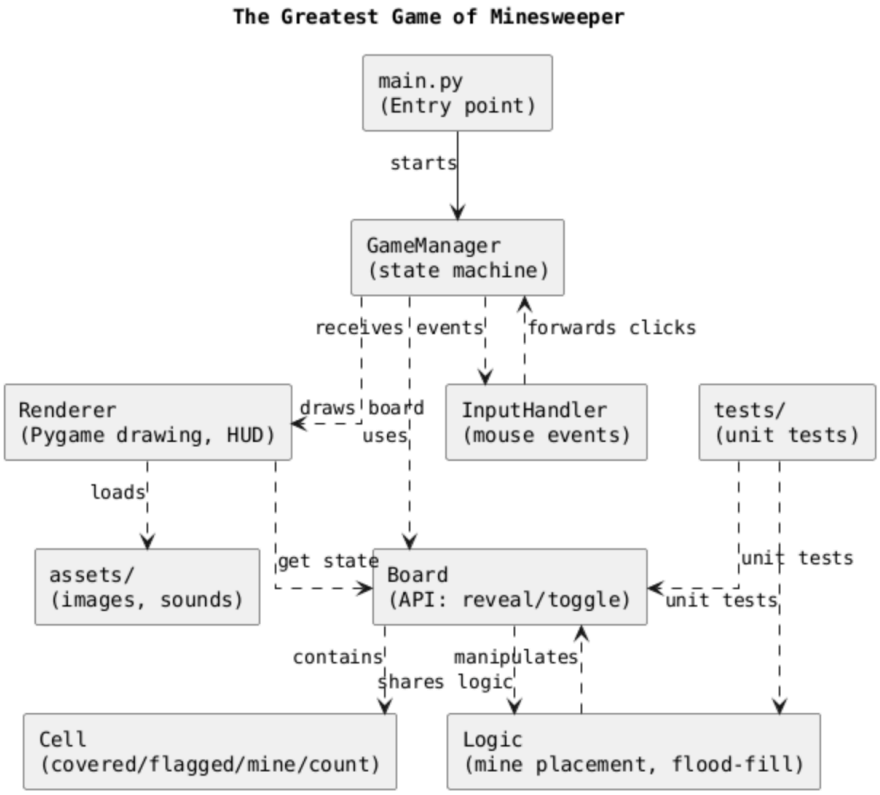
# 4. Logical View

## 4.1 Overview

The overall design of the Minesweeper program is divided into five major subsystems that interact to deliver the game's functionality. These are the

- **Game Manager**: acts as the central controller, orchestrating interactions between components, managing states (PLAYING, WIN, LOSS), and handling the game loop
- **Board & Logi**c: both are responsible for mine placement, recursive uncovering (flood fill), and win/loss detection
- **Data Structures** (Cells/Board Classes): define the grid of Cells and Board API, storing essential game state information (mines, flags, covered/uncovered)
- **User Input Handle**r: processes mouse and keyboard inputs, mapping them to board actions (reveal cell, toggle flag, restart)
- **Renderer**: dynamically draws the board and UI using Pygame, showing cell states, flags, mines, counters, and banners.

## 4.2 Architecturally Significant Design Modules or Packages



# 5. Interface Description

The Greatest Game of Minesweeper interface will be built with Pygame, with a grid based design as per the EECS 581 project requirements:

- Valid Inputs:
  - Left click: Reveal a cell
  - Right click: Place/remove a flag
  - R Key on keyboard: Reset the board for a new game
  - Esc Key: Exit game
- Outputs:
  - Rendering of board updates (reveals, flags, mines)
  - Game status text (“Playing”, “Victory!”, “Game Over”)
- Data Storage:
  - Game state (board + cells) is stored in-memory within the Board object
  - No external persistence is required

# 6. Development Description

## 6.1 Overview

The Greatest Game of Minesweeper was built following an Agile system with scrums lasting a week. Every Monday, we met with our scrum master and discussed upcoming and completed work. The expected and actual times for the work are as follows:

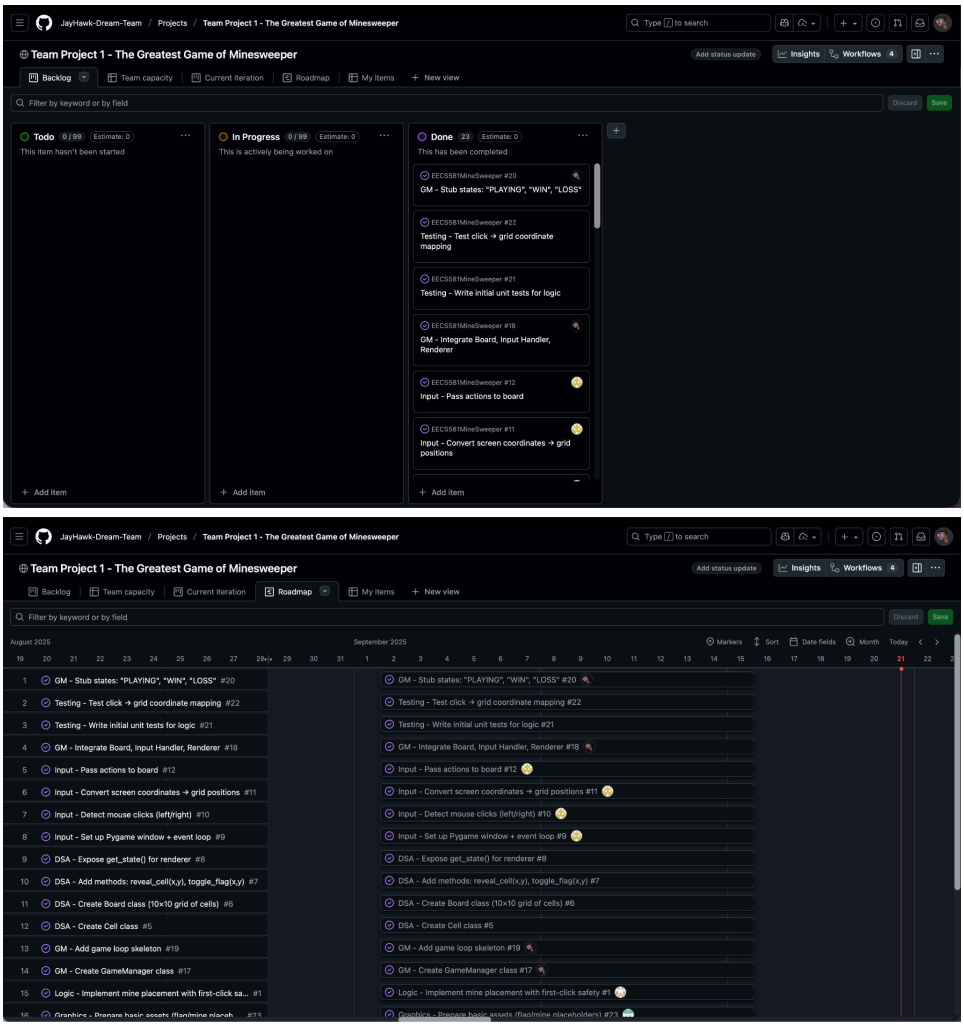
Carlos Mbendera: Expected 2 Hours, Took 3 Hours
Jonathan Johnston: Expected 4 Hours, Took 3 Hours and 30 Minutes
Yoseph Ephrem: Expected 3 Hours and 30 Minutes, Took 3 Hours
Cole DuBois: Expected 3 Hours, Took 2 Hours
Blake Ebner: Expected 2 Hours, Took 45 minutes
Mahdi Essawi: Expected 3 Hours, Took 3 Hours

These times exclude meetings and research.

To calculate the expected amount of time for work, we allowed each individual to approximate their own respective hours as we believed that due our own personal difference in experience and technical history that we’d be our best estimates since the workload was divided depending on what everyone’s strengths were. For example, Blake worked on Rendering the MineSweeper board and has a history in game development via personal projects while the rest of the team had little to no experience in that tech stack. Thus, we trusted his estimate on the work..

Moreover, during the development of The Greatest Game of Minesweeper, we used GitHub extensively and GitHub Projects to keep track of our roles, tasks and timelines of each other's work. Our repository is here and the project is here.

By observing our GitHub Project, you shall notice that we had a planning board to separate tasks into 3 sections, Todo, In Progress, Done. This made it possible for everyone to be synchronized on the status of components they may be dependent on. Additionally, we also had used the Roadmap feature to get a rough timeline of when everything was needed on an interaction by iteration basis. However, we spent the first iteration planning and distributing tasks, so we only coded for a single iteration. Hence, our Roadmap only has one iteration. Below are screenshots of the GitHub Project showing both pages..







## 6.2 Meeting Notes

**September 2, 2025**
Initial Meeting following the creation of teams and assignment of the project. This meeting mainly focused on understanding the code required to build minesweeper, learning the team's strengths and how we may distribute tasks. Following the meeting, we agreed to use Python with Pygame, perform research and create a shared repository and GitHub Project for organization. The tasks were assigned as follows:
1. Board & Mine Generation (Logic) - Jonathan
2. Cell & Board Classes (Data Structures) - Yoseph And/OR Mahdi
3. User Input Handling (Controls) - Cole
4. Graphics / Rendering - Blake
5. Game Manager (Glue + State Machine) - Carlos
6. Testing / Polish / Extras - Yoseph And/OR Mahdi

Yoseph and Mahdi worked as a small mini team and could choose how to share tasks between themselves.

**September 5 and 8, 2025**
These two meetings represent our first meetings with our TA (Scrum Master) whilst we were scheduled to meet on Mondays, Labor day caused us to meet our TA on Friday 5 September and again on Monday 8 September. Due to the quick turn around, I've combined the notes for both meetings as this was still our Research phase and we synced our TA on how we're planning to work moving forward and they communicated with us further requirements and grading criteria for the project.

**September 15, 2025**
During this Scrum meeting with our TA we had developed most of the core functionality and our remaining tasks were related to integration of the decoupled modules together, testing and polish. Communication between the team outside meetings was mainly through GroupMe messages in a group chat.

**September 19, 2025**
We had a short and casual meeting discussing the project and who was going to submit the documentation and github link. We also experienced a quick demo with the new polish added by the members working on quality.

## 7. Quality

The quality assurance process for the minesweeper project consisted of debugging, unit testing, and performance goal testing to ensure an enjoyable gameplay experience for our users. Our initial rounds of the manual testing uncovered a few bugs. These included input options for the mine count, misaligned board labels, and a missing flag counter feature. These errors were assessed promptly and corrected, improving the ease of accessible information and customization of the game.

After fixing these issues we designed a unit testing script with the help of AI to validate every feature, class, and interaction in our code. This script systematically tested the game's features piece by piece, navigating through the various connections and wiring. The script covered both functional and edge case scenarios to minimize unexpected behavior from the game. A total of 32 unit tests were created and all passed successfully, showcasing that our system logic and feature set was operating as intended.

Beyond correctness, we also prioritized efficiency by embedding performance benchmarks directly into the test script. These benchmarks measured our execution time and responsiveness for critical operations in the game including:

- Small board creation
- Large board creation
- First click (small board)
- First click (large board)
- Flood reveal

Each benchmark was evaluated under strict conditions, allowing no room for error, to ensure that the gameplay was smooth and consistent regardless of the board's size or difficulty. The results were highly favorable, with all benchmarks passing comfortably within their defined performance thresholds. This confirmed that the game engine could handle both small and large configurations without compromising the responsiveness or stability of the game.