Project: Co-op Sudoku

Matthew Davis, Ethan Tran, and Cole Hyink

CIS 350: Fall 2020

Table of Contents

Project Information

This program should be able to play Sudoku in a new way! With this app we plan to allow competitive sudoku play to be hosted on our server. Two players with the same link are given the same board and the time played is the measure of skill between all players.

Features

Included in this release is a functioning Sudoku game. The player is able to choose between three difficulty settings: Easy, Medium, and Hard. The player is able to make turns to complete the board, as well as undo turns. You can give up on the current board you have. If you win a game, a message is displayed declaring you won. From there you are prompted to select a new difficulty and play a new game.

Screenshots

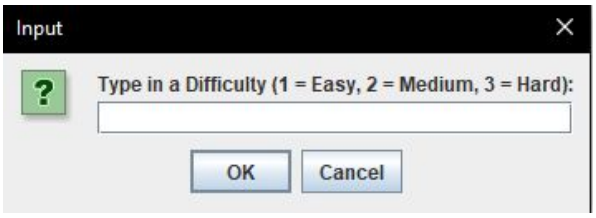

Figure 1. Difficulty User Interface

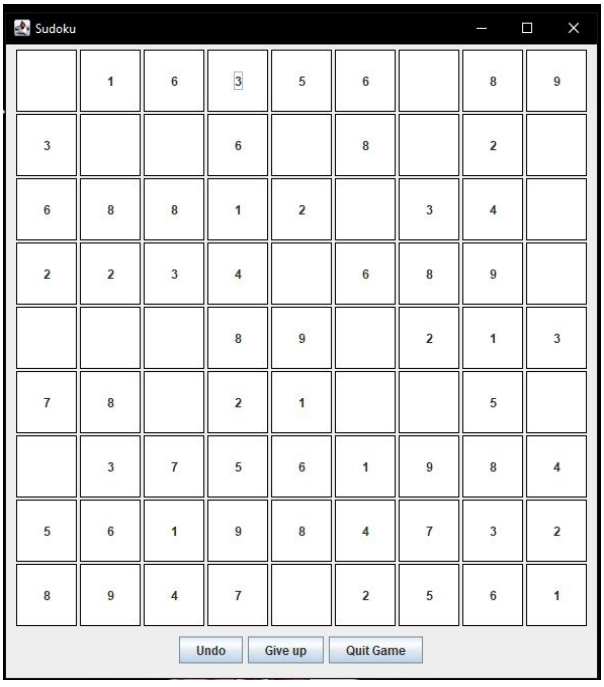

Figure 2. Sudoku Board User Interface

Project Plan

Our project will follow the Waterfall Model. Our resources will be a free server to host the website. Our tasks include hosting the app on a website, as well as creating and implementing the leaderboard and database. There are no risks associated with the project, and we plan to have the app playable by 10/19/20.

Requirements & Definition

Business Requirements
- Play Sudoku

User Requirements
- Login and registration
- Be able to play the game
- Have an undo button

Functional Requirements
- Login authentication
- Log games won and given up on
- Server security
- App is functional

Non-Functional Requirements
- User friendly and easy to use
- Save Gamestate incase of power cut

Inception
- Online Sudoku that tracks the amount of games won, amount of games given up
- Storing of games won and given up allows for competition among friends

Elicitation
- Goals: develop a sudoku application with leaderboards
- User classes: Player, board
- Priority: Game works, and the leaderboard is set up
- Architecture: Java
- Normal requirements: Login, leadboards, filter
- Expected requirements: Fast, easy to use, good looking?
- Exciting requirements: Co-op

Elaboration
- ~~Login Use Case~~
  - ~~Use Case Name: Login~~
  - ~~Primary Actor: Player~~
  - ~~Goal: Let player login~~
  - ~~Preconditions: System can accept username and password, database is built~~
  - ~~Trigger: Player clicks login button~~
  - ~~Scenario: Player clicks on app, clicks login button~~

    - ~~Exception: invalid login, invalid username, invalid password, servers down~~
- Leaderboard Use Case
    - Use Case Name: Leaderboard
    - Primary Actor: Player
    - Goal: Display leaderboard to player
    - Preconditions: Database is built and populated
    - Trigger: Player clicks leaderboard button
    - Scenario: Player clicks on app, clicks leaderboard button
    - Exception: invalid login, servers down, database does not work
- Game Use Case
    - Use Case Name: Game
    - Primary Actor: Player
    - Goal: Let player play the game
    - Preconditions: Login and registration
    - Trigger: Player clicks play button
    - Scenario: Player clicks on app, clicks on play button
    - Exception: Invalid login, servers down

Negotiation

- Co-op
    - Users may want to face off against their friends in real time, but we may not have the experience or expertise to code that. Instead, we can create a shareable link for the specific board generated by one of the players and they play the board while their time and or number of moves is recorded
- Server
    - We have no experience with serve coding, so we shouldn't have server aspects to the project

Specification

- System Requirements
- External Interface Requirements
- Non-functional Requirements

Validation

- Prototypes
    - 1st prototype is a basic screen that displays a sudoku board
    - 2nd prototype allows for the generation of a sudoku board
    - 3rd prototype allows for playing of sudoku
    - 4th prototype difficulty settings

Management

- Requirement tracking as we complete project

Development

Create simple game logic to solve and play sudoku, along with a graphical user interface to enhance the player experience. Implement a leaderboard system to allow players to access their wins and losses.

Verification

To verify that the project is complete, we will assess the program for any bugs, make sure that the graphical user interface is subjectively good, the leaderboard is functional and operates well with the database, the game logic is functional, and the interaction with buttons are functional.

Maintenance

- Identify problem
    - Label Problem
    - Track all problems in a file
    - Problem discovery date, confirmed removal of problem date, description of problem, how to replicate problem, how problem was solved
- Post release, link to submit problems
    - Ask for description of problem, how problem occurred, ask if they can explain how to repeat problem

Umbrella Activities

[Plan for project management and status tracking / meetings detailed here]

Meetings will occur on Mondays and Fridays at 9pm, and Wednesdays at 9:30PM. Additionally, Tuesdays and Thursdays from 4-5pm will be additional meeting times if needed.

Responsibilities

We all share equal responsibilities for creating code, updating documentation, and presenting the project. The work will be split when we are in meetings. There will be a rotation of someone doing coding and two people working on the documentation. This seemed to be the most efficient way to code because if the person coding gets stuck someone can still help him while work is still being done on the documentation. Somehow if we can, we could possibly have two people coding on the same document simultaneously.

Requirements & Specification

The player can currently launch the application and start the game with a set difficulty. The buttons on the board can be pressed and updated and calculate the game state using the game logic. There is an undo button which undoes the last board pressed and there is also a give up button which allows the player to look at the game logic visually. Further implementation will include automatic solving and displaying of the board.

Use-Cases

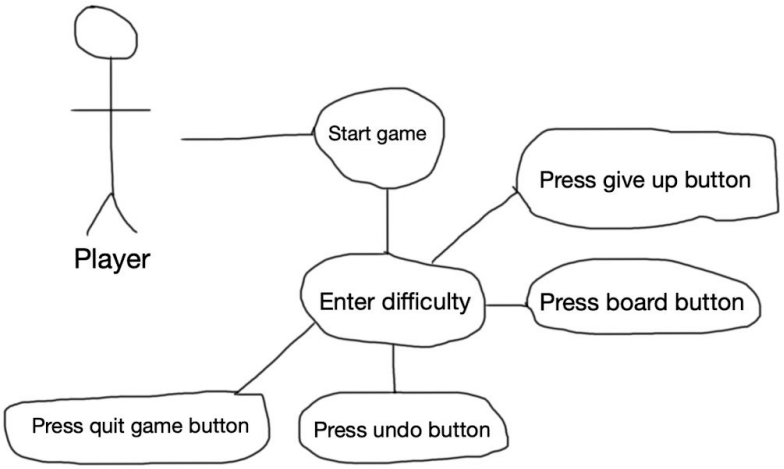

Figure 3. Use Case Diagram

As of now, the use case for playing the game is when the application launches, the user is prompted to enter in the select difficulty with a backdoor key of “666” to automatically generate and solve a board. The user will then play the game until completion, then is prompted with “You win” and is asked to play another game. The methods used in this game consist of mainly game logic methods which determine the game state of the board and also the legality of the move the player had made. There is also a use case where if the player accidentally clicked a button they did not intend on clicking they may click the undo button and the last button pressed will be unpressed. The undo button is a stack of positions which can be used to undo the game down to the very beginning.

Traceability Matrix

REQUIREMENTS TRACEABILITY MATRIX					
Project Name: Online Sudoku					
Business Requirements Document BRD		Functional Requirements Document FSD			Test Case Document
Business Requirement ID#	Business Requirement / Business Use Case	Functional Requirement ID#	Functional Requirement / Use Case	Priority	Test Case ID#
BR_1	~~Login Module~~	FR_1	Login Authentication	High	TC#001 TC#002
BR_2	Play Module	FR_2	Display Board	High	TC#003
		FR_3	By Difficulty	Medium	TC#004
BR_3	~~Leaderboard Module~~	FR_4	Log wins/losses	Medium	TC#005 TC#006
		FR_5	Display Leaderboard	High	TC#007

Figure 4: Requirements Traceability Matrix

Design

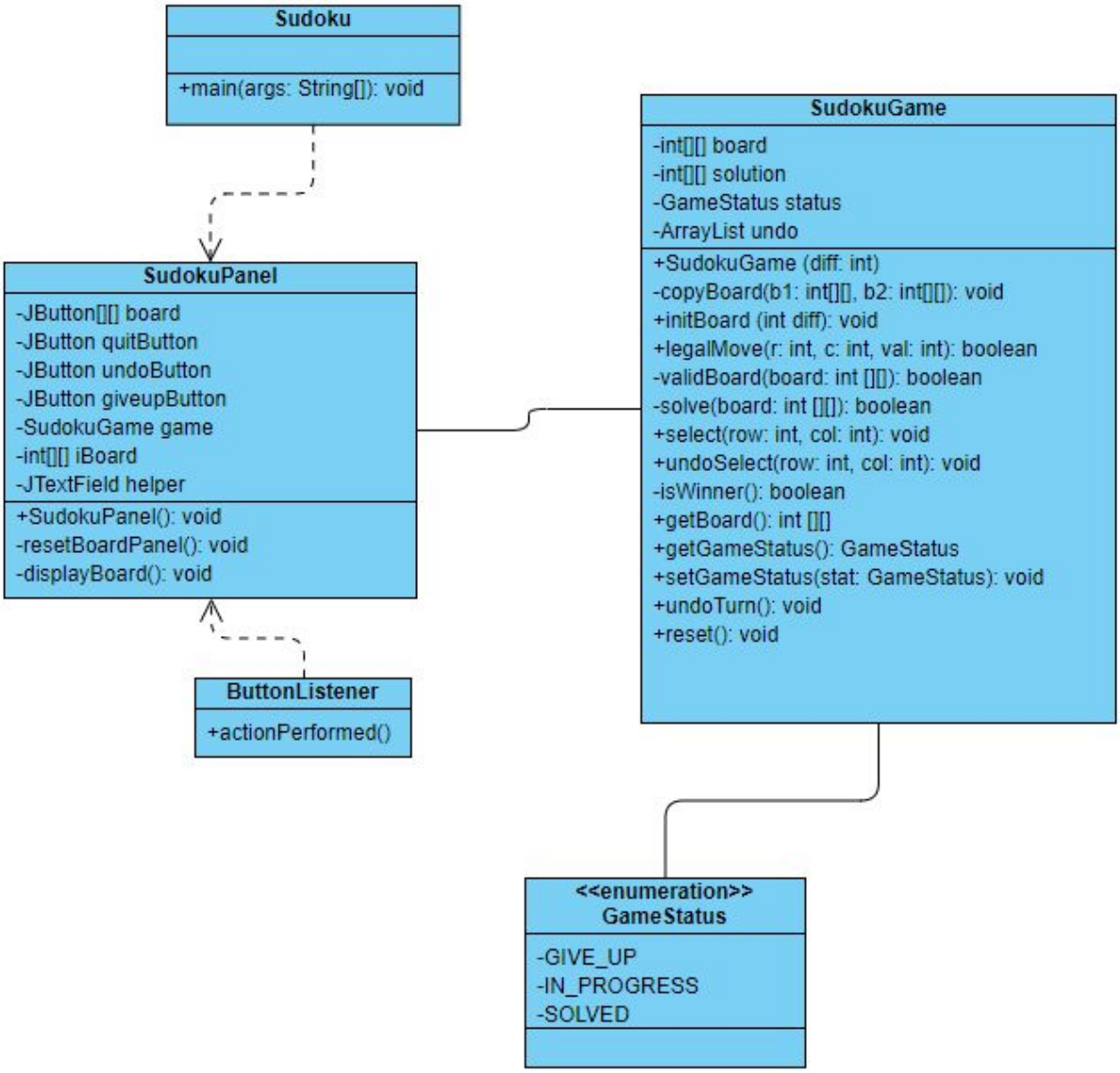

Figure 5: Sudoku Game Class Diagram

GameStatus.java

Defines the three different states of the game, “IN_PROGRESS”, “SOLVED”, and “GIVE_UP” using an enumerated type.

Sudoku.java
Creates the application screen. A JFrame is created and the default close operation is set to EXIT_ON_CLOSE. The SudokuPanel class is called to create a new panel for the JFrame and is added to JFrame.

SodkuGame.java
Creates and verifies that the board and any moves are valid. A board consists of a 9 by 9 integer array. The constructors sets the GameStatus to IN_PROGRESS, and then creates a board and solution array that are both 9 by 9. The reset() method is called to clear the board the user will see. For the default constructor, the board is initialized by calling the initBoard(int diff) method with a diff of 1. For SodokuGame(int diff) constructor, the initBoard(int diff) command is called and given the value of diff. The class also has methods for copying the board to another board, determining if a move is legal, determining if the board is valid, selecting a certain row and column of the board, undoing a selection, determining if the board is a winner, and undoing terms. It has getters methods for the board variable and GameStatus, as well as a setter method for the GameStatus.

SudokuPanel.java
Creates JButton’s and JTextField’s for the SudokuGame. Buttons are set for quitting the game, undoing a move, and giving up on the current game. A method can be called to display the board, as well as reset the board panel. Button listeners are implemented for each button to undo a turn, give up on the game, make a turn, as well as check if the game has been won.

Development

Arraylist, Java Swing, Java awt. The use of Arraylists in this project is to allow ease of the undo button to be implemented. Java Swing had been used as a user interface. Java awt had been used to implement the buttons in the interface. For the final release, we plan

Code Standards

For the project we used the Checkstyle to keep our code to standard. We adjusted the code to keep it aligned with acceptable Checkstyle standards after initial writing of the code.

Static Analysis

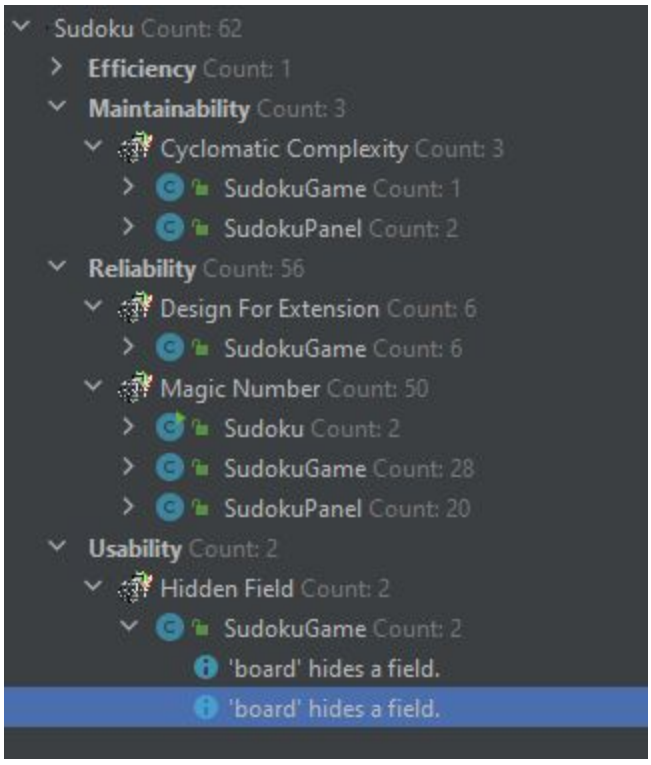

Figure 6: Find Bugs

Figure 7: CheckStyle

Code Documentation

For the first release, we were not able to complete javadocs. When preparing release one, we realized we needed to focus our efforts on more important parts of the release. We plan on having the javadocs completed for the final release, with current plans to create a javadoc for the current code after the release is finished and turned in.

Configuration Management

https://github.com/Ekinsxd/CIS350Project

For tracking our releases, we will use the release dates set for the class for the initial releases. For future releases, github will be important to tracking the releases by developing new features on new branches and only adding to master when we are confident they will work.

Verification

[Description of methods used (e.g., integration & systems and/or unit testing)]

Integration Tests

[Include manual and integration test procedures]

Unit Tests

[Include references to unit tests in code]

None yet

Code Coverage

[Include code coverage reports, must include: coverage of automated tests, coverage of manual tests, and combined coverage]

Requirements Coverage

[Include traceability (matrix) from requirements to test procedures]

Postmortem

So far the project has gone alright. Meetings have been productive and the project was always discussed or worked on each meeting. If a member knew they were going to miss the meeting, they were able to let the group know beforehand. As far as the project, we were forced to change our project topic before the first release due to lack of communication from a key group member who had the most knowledge on the project subject, which left the rest out of our depth in terms of coding for that project. As a result, we switched to a Sudoku game with additional features.

Earned Value

[Include the earned value calculations for your current status and any explanation of over/under runs]

Our project time budget is 14 weeks. First release has used 7 weeks, which has produced the working Sudoku game. Working forward we have 7 more weeks of production before the final release, in which we need to produce a leaderboard, database, and site to host the game. We encountered difficulties with our previous project idea, where the project was beyond our understanding. After the loss of a group member, we decided to pivot projects so we could finish the product on time. This left our group with little time to produce a Sudoku game before release 1, explaining why we do not have a leaderboard, database, or website.

Variances

[Include any additional variance (time, coverage, functionality, …) explanations necessary]

Lessons Learned

Sticking close to a clear plan, starting early to provide enough time for error correction, documentation consistent through time frame.

References

[Include references here. Use *consistent* citation method.]