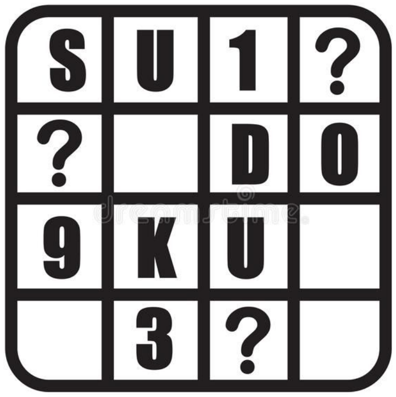
Project: Sudoku



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.

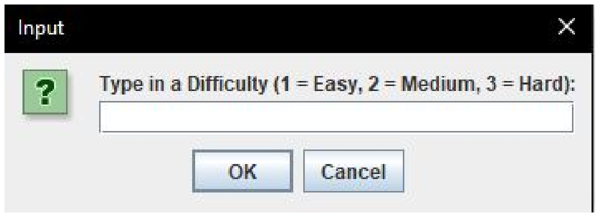
Features

The program will feature a functioning Sudoku game. The starting board can have a difficulty set by the player that is either easy, medium, and hard. The board will be randomized each time to present the player a unique board every new game.

Players will be able to select a square and input a number. This turn is saved, and is able to be undone at a later point if the player deems necessary. The board will also feature a hint button, which will show the spots on the board that are currently correct in terms of creating a solvable Sudoku board, as well as the spots on the board that are preventing the board from being solvable. If the player is unable to solve the board, a give up button is available to complete the board. Players will also have an option to quit the game.

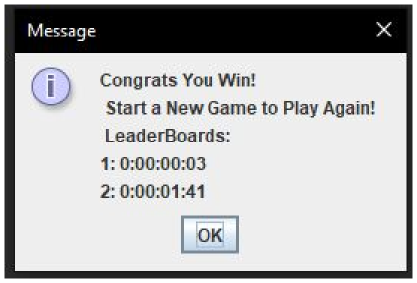
In addition to the board features, a clock will keep a running time that the player has spent solving the current board and display it. Games will also be saved after every turn in case of computer or program failure. A leaderboard tracking the top ten best times it took to complete a Sudoku game was also tracked.

Screenshots



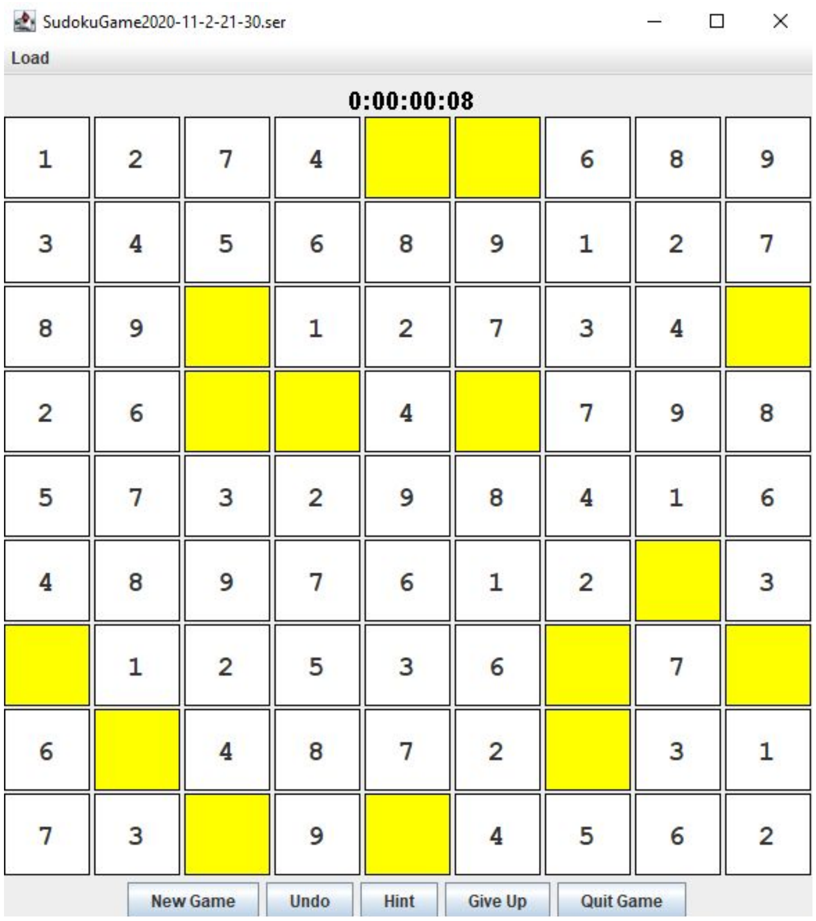
Type in a Difficulty (1 = Easy, 2 = Medium, 3 = Hard):

Figure 1. Difficulty User Interface



Congrats You Win!
Start a New Game to Play Again!
LeaderBoards:
1: 0:00:00:03
2: 0:00:01:41

Figure 2: LeaderBoards User Interface



1	2	7	4			6	8	9
3	4	5	6	8	9	1	2	7
8	9		1	2	7	3	4	
2	6			4		7	9	8
5	7	3	2	9	8	4	1	6
4	8	9	7	6	1	2		3
	1	2	5	3	6		7	
6		4	8	7	2		3	1
7	3		9		4	5	6	2

Figure 3. Sudoku Board User Interface

Project Plan

Our project will follow an Agile Model. Our resources will be the java api mainly java awt and java swing. These two development tools will be the main tool kits being used to create the user interface and logic for the sudoku game. There are no risks associated with the project, and we plan to have the app playable by 10/19/20. Once the app is playable, we will then add additional features based on the importance of the feature in regards to the player. For example, getting a Sudoku game that automatically saves after every term will help the player if the game unexpectedly quits. We also will aim to have a timer implemented to keep track of how long it takes for the user to complete the current board. A leaderboard of the top ten fastest board solves will be kept and displayed after a board has been completed.

Requirements & Definition

Business Requirements
- Play Sudoku

User Requirements
- Edit the game board
- Have an undo button
- Have a hint button
- Have an Give up button
- Have a quit button
- Have a new game button
- Have a load button

Functional Requirements
- Leaderboard
- App is functional

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

Inception
- Sudoku that tracks the top ten fastest times that the user had solved the board
- Storing of game times allows for improvement
- Hint button to help players struggling to solve board
- Undo button for easy retraction of mistakes
- Give up button to show a completed board for the game

Elicitation
- Goals: develop a sudoku application with leaderboards
- User classes: Panel, Game
- Priority: Game works, and the leaderboard is set up
- Architecture: Java
- Normal requirements: leaderboards, Autosave
- Expected requirements: Fast, easy to use, good looking?

Elaboration
- Leaderboard Use Case
  - Use Case Name: Leaderboard
  - Primary Actor: Player
  - Goal: Display leaderboard to player
  - Preconditions: Leaderboard is built and populated
  - Trigger: Player completes sudoku board
  - Scenario: Player completes the sudoku board
- Board Use Case
  - Use Case Name: Board
  - Primary Actor: Player
  - Goal: Display board to player

  - Preconditions: Player has selected difficulty
  - Trigger: Player selects difficulty
  - Scenario: Player runs game, selects difficulty
  - Exception: Difficulty too large, or incorrect type
  -
- Update Board Use Case
  - Use Case Name: Update Board
  - Primary Actor: Player
  - Goal: Update board to player
  - Preconditions: Player has started the game
  - Trigger: Player presses enter after entering a key into textfield
  - Scenario: Player runs game, selects difficulty
  - Exception: Difficulty too large, or incorrect type
- Game Use Case
  - Use Case Name: Game
  - Primary Actor: Player
  - Goal: Let player play the game
  - Preconditions: Launched game
  - Trigger: Player clicks play button
  - Scenario: Player launches app and selects difficulty

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 implemented a leaderboard in which you can practice and try to complete the sudoku board as fast as possible.

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
    - 5th prototype fixed bugs
    - 6th prototype added save/load
    - 7th prototype added timer
    - 8th prototype fix bugs in timer and load/save
    - 9th prototype add leaderboard and fix misc. bugs

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

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.

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

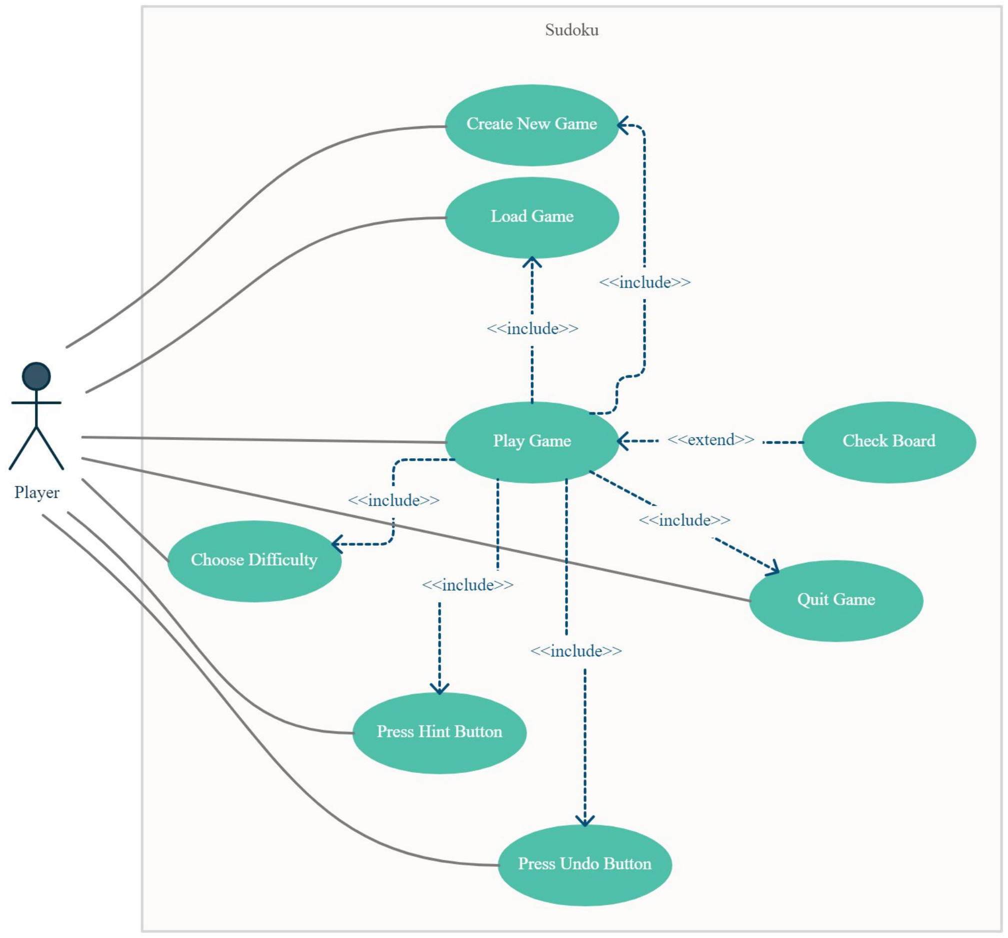


Figure 4. Use Case Diagram

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		
Business Requirement ID#	Business Requirement / Business Use Case	Functional Requirement ID#	Functional Requirement / Use Case	Priority
BR_1	Board Module	FR_1	Create Board	High
BR_2		FR_2	Display Board	High
	Play Module	FR_3	Easy Difficulty	High
		FR_4	Medium Difficulty	Medium
		FR_5	Hard Difficulty	Medium
BR_3	Save Module	FR_6	Save Board	Low
		FR_7	Load Board	Low
BR_4	Timer Module	FR_8	Display Time	Medium

Figure 5: Requirements Traceability Matrix

Design

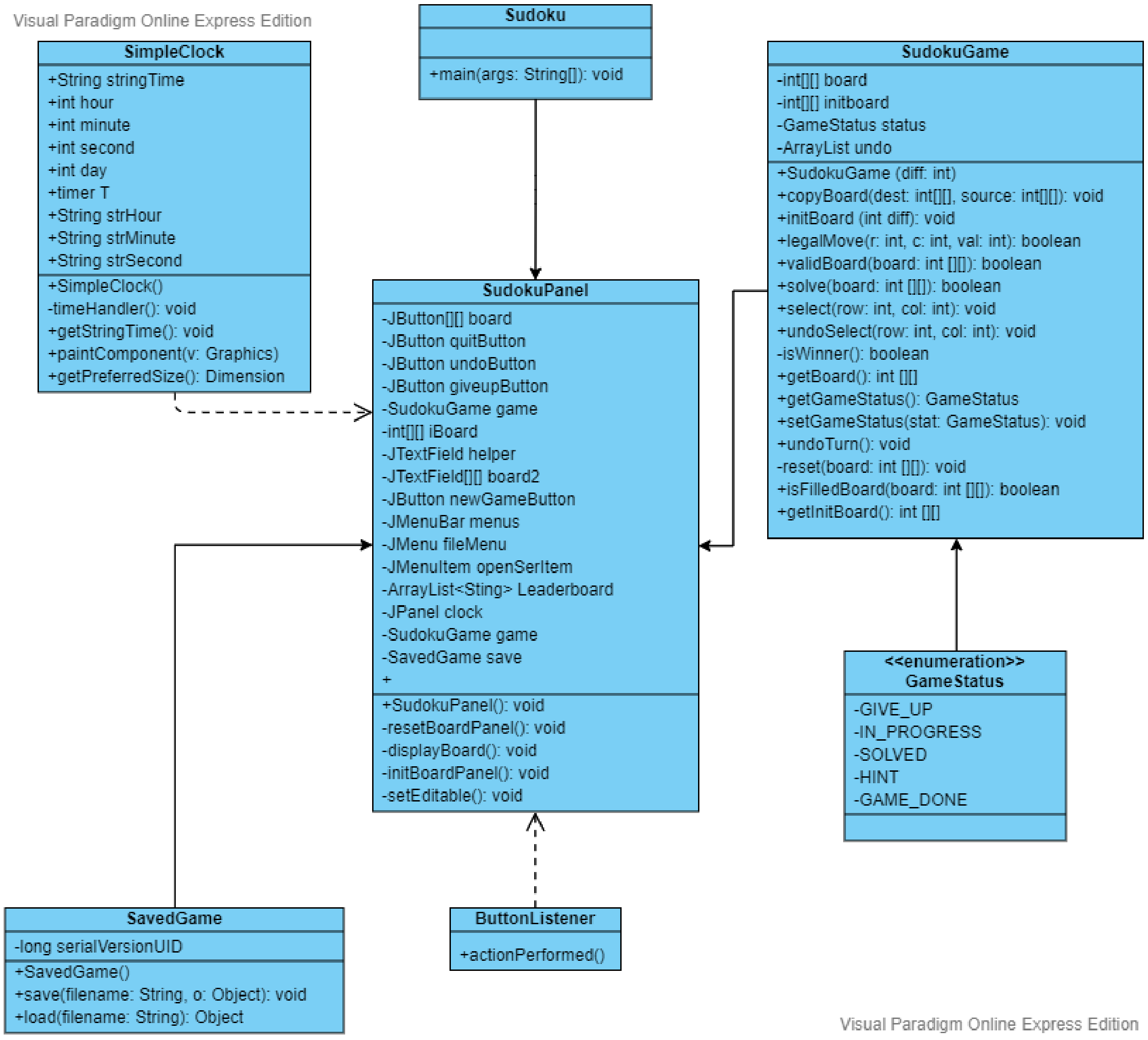


Figure 6: Sudoku Game Class Diagram

GameStatus.java
Defines the three different states of the game, “IN_PROGRESS”, “SOLVED”, “HINT”, “GAME_DONE”, 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 getter methods for the board variable and GameStatus, as well as a setter method for the GameStatus.

SudokuPanel.java
Creates JButtons and JTextFields 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.

SavedGame.java
Adds the ability to save and load games through serializing objects utilizing the interface serializable.

Clock.java
Class that creates and manages the clock that runs to track the amount of time spent on a game. Has methods that can stop the timer, restart the timer, reset the clock, and update the time on the GUI.

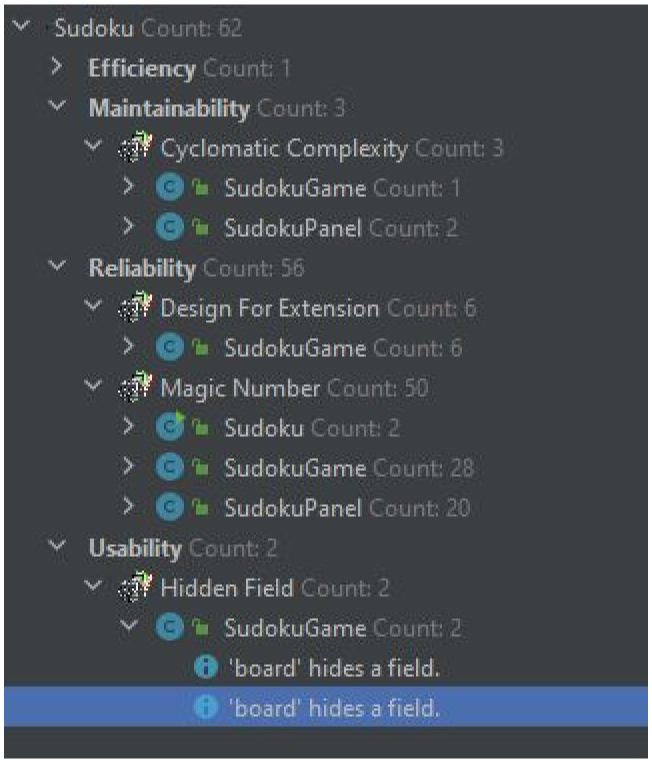
Development

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 to use a facade pattern to hide the game logic from the user and present the code as one panel containing everything. The user must simply use input to play the game and nothing else may be touched by the user.

Code Standards

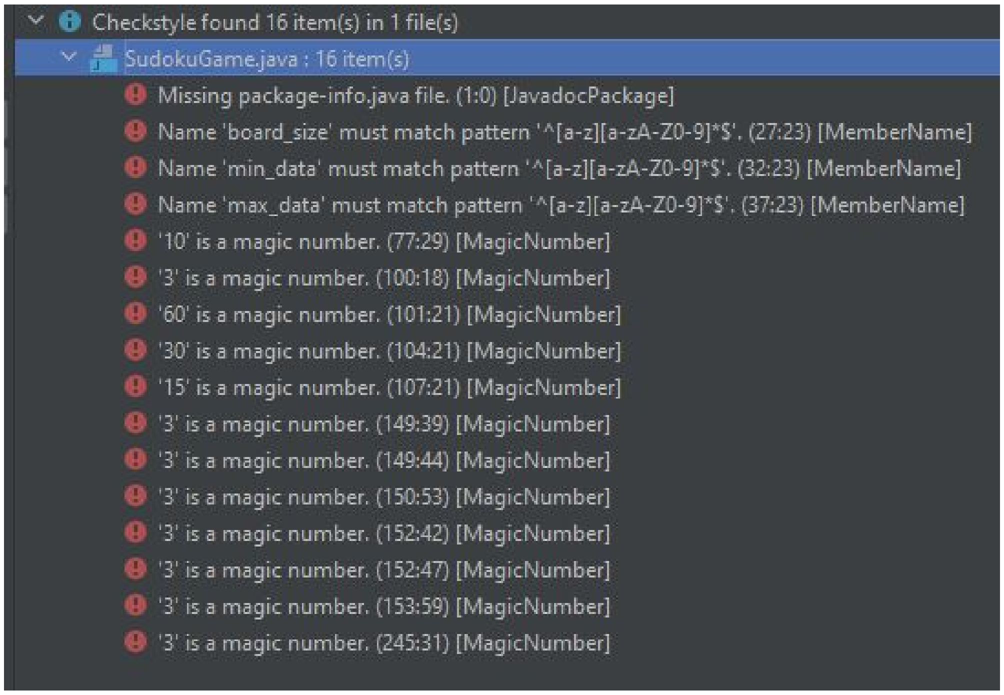
For the project we used the Checkstyle to keep our code to standard. We adjusted the code to keep it aligned with acceptable Checkstyle standards after initial writing of the code. We used JavaDocs as a basis to comment on our code.

Static Analysis



```
Sudoku Count: 62
  Efficiency Count: 1
  Maintainability Count: 3
    Cyclomatic Complexity Count: 3
      SudokuGame Count: 1
      SudokuPanel Count: 2
  Reliability Count: 56
    Design For Extension Count: 6
      SudokuGame Count: 6
    Magic Number Count: 50
      Sudoku Count: 2
      SudokuGame Count: 28
      SudokuPanel Count: 20
  Usability Count: 2
    Hidden Field Count: 2
      SudokuGame Count: 2
        'board' hides a field.
        'board' hides a field.
```

Figure 7: Find Bugs



```
Checkstyle found 16 item(s) in 1 file(s)
  SudokuGame.java : 16 item(s)
    Missing package-info.java file. (1:0) [JavadocPackage]
    Name 'board_size' must match pattern '^[a-z][a-zA-Z0-9]*$'. (27:23) [MemberName]
    Name 'min_data' must match pattern '^[a-z][a-zA-Z0-9]*$'. (32:23) [MemberName]
    Name 'max_data' must match pattern '^[a-z][a-zA-Z0-9]*$'. (37:23) [MemberName]
    '10' is a magic number. (77:29) [MagicNumber]
    '3' is a magic number. (100:18) [MagicNumber]
    '60' is a magic number. (101:21) [MagicNumber]
    '30' is a magic number. (104:21) [MagicNumber]
    '15' is a magic number. (107:21) [MagicNumber]
    '3' is a magic number. (149:39) [MagicNumber]
    '3' is a magic number. (149:44) [MagicNumber]
    '3' is a magic number. (150:53) [MagicNumber]
    '3' is a magic number. (152:42) [MagicNumber]
    '3' is a magic number. (152:47) [MagicNumber]
    '3' is a magic number. (153:59) [MagicNumber]
    '3' is a magic number. (245:31) [MagicNumber]
```

Figure 8: 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. Preparing for the final release, we have completed javadocs, while including an analysis on bugs, and CheckStyle. Our analysis found 62 potential bugs across the game, while our CheckStyle found 16 items that do not fit code standards.

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.

Person	Ethan Tran	Cole Hyink	Matthew Davis
Number of Commits	27	7	9

Figure 9: Number of Github Commits

Verification

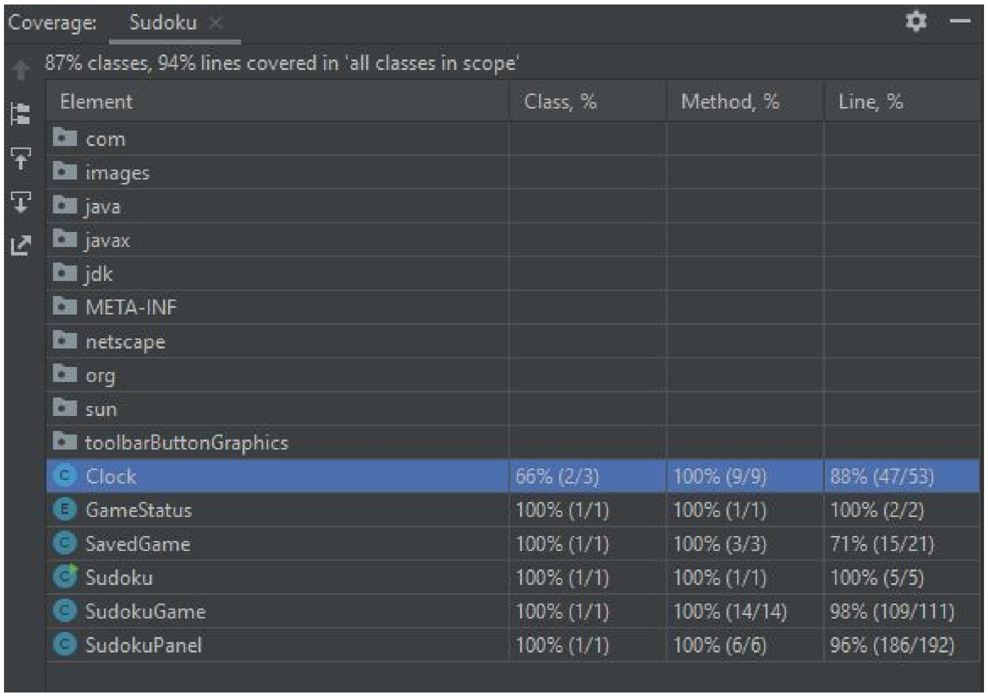
We used validation testing to ensure that the Sudoku game is playable and all features we wanted to implement were functional. We used IntelliJ code coverage to test if our code is being used along with hand testing the .

Unit Tests

Very hard to do unit test cases for a game. The UI had been rigorously hand tested and any bug encountered had been fixed. There are many cases to ensure that the program does not give red lines or cause an error.

Using the UI, testing the code had been done by using the interface and ensuring that the no invalid input could be entered into the board with any errors or illegal moves. For example, no negative numbers, no numbers not within 1-9, no letters, no super long strings of numbers, and so on. The buttons had been tested to work in tandem with one another. For example, while the hint button is activated all other buttons are still functional. If the undo had been pressed, the hint button acknowledges that the hint is active and the hint functionality still works. Most all bugs from the user interface have been handled through alpha testing and works to the best of our ability. The save and load functionalities are in good working order, although they could be expanded upon to improve the user experience, the project is complete for now, and this can be handled in postmortem where future work may be done.

Code Coverage



Coverage: Sudoku

87% classes, 94% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
com			
images			
java			
javax			
jdk			
META-INF			
netscape			
org			
sun			
toolbarButtonGraphics			
Clock	66% (2/3)	100% (9/9)	88% (47/53)
GameStatus	100% (1/1)	100% (1/1)	100% (2/2)
SavedGame	100% (1/1)	100% (3/3)	71% (15/21)
Sudoku	100% (1/1)	100% (1/1)	100% (5/5)
SudokuGame	100% (1/1)	100% (14/14)	98% (109/111)
SudokuPanel	100% (1/1)	100% (6/6)	96% (186/192)

Figure 10: Code Coverage

The line coverage for the project when ran, as if a user is normally playing it, does not have 100% line coverage through every class due to try catches in classes not catching any exceptions when the code is run.

Requirements Traceability Matrix

REQUIREMENTS TRACEABILITY MATRIX				
Project Name: Online Sudoku				
Business Requirements Document BRD		Functional Requirements Document FSD		
Business Requirement ID#	Business Requirement / Business Use Case	Functional Requirement ID#	Functional Requirement / Use Case	Priority
BR_1	Board Module	FR_1	Create Board	High
BR_2		FR_2	Display Board	High
	Play Module	FR_3	Easy Difficulty	High
		FR_4	Medium Difficulty	Medium
		FR_5	Hard Difficulty	Medium
BR_3	Save Module	FR_6	Save Board	Low
		FR_7	Load Board	Low
BR_4	Timer Module	FR_8	Display Time	Medium

Figure 11: Requirements Traceability Matrix

Customer Acceptance Criteria

The project encompasses all of the requirements set forth from the beginning. We began focusing our efforts on making a working Sudoku game with an undo button. This was accomplished so that the game is user friendly and functional. Once user requirements were completed, we created save functionality to satisfy non-functional requirements, and later we proceeded with a leaderboard to finish functional requirements.

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.

Matt Project Reflection

Overall, I felt this project went along alright. The beginning was a little rough due to one member dropping the class, but I feel the rest of our group pulled together well to move forward. As far as personal reflection, I feel I could have done more to help the project. I was to do the documentation and implement the JavaDocs for the code. I feel at points I slipped in keeping up with doing my role, that I could have done better. Another role I had for the group was to set up meetings and ensure that everyone knew that they were happening.

I feel the project went well though. Cole and Ethan worked hard on the code and got major features implemented into the project. The project has come a long way since our first meeting and even our first project release. Overall I think we worked well as a team.

Cole Project Reflection

I felt as though our project went well for starting late and having one less member. The beginning was overwhelming as I am behind on my Java experience, but after a while I began to settle into my role. Because of this, it took more time than anticipated to complete my tasks, but after some assistance the code came together well. Overall, I felt the project went well for how much time and experience we had. The project helped me learn the process of team coding.

Ethan Project Reflection

This project has been difficult as we had a group member leave us halfway through the project while we didn’t have any work done. Although we had to go through that, the product that we had created had been satisfactory. I believe that the game that we created had been good for the amount of time and effort we put into this project. Overall, this project had been a good insight into the software engineering process.

Responsibilities

Ethan's main priorities were to create a functioning undo button, hint button, and have the game randomizer functioning. Once he completed his main priorities, he assisted in getting the leaderboard, timer, and save game functionalities to work.

Cole's main priorities were to create the save game functionality, as well as the timer function. After that, he worked on fixing bugs and errors in the code and creating a functioning leaderboard for the game.

Matthew's main priorities were to create the required documentation for the project and ensure the code had proper JavaDocs implementation.

Earned Value

Our project time budget is 14 weeks. First release has used 7 weeks, which has produced the working Sudoku game. Working forward another 7 weeks produced saved game functionality, load game functionality, leaderboard, and timer. Attempted but failed ideas include database, login, better GUI, server, web-hosted game.

Lessons Learned

Over the course of the project we learned a few key lessons. One of the main lessons learned was the importance of communication. Throughout the project, there were times where we miscommunicated what needed to be done or what was done. This led to confusion and used up time that could have been spent elsewhere in the project. Even early on, there was confusion between if a member was still a part of the project, which caused us headaches as we tried to complete a project we had very little knowledge on. Another important lesson learned was the importance of a schedule and sticking to it. We were sidetracked easily throughout the project and lost sight of our goals. We didn't work on the project as much as we could have, which reflects in the quality of the project.