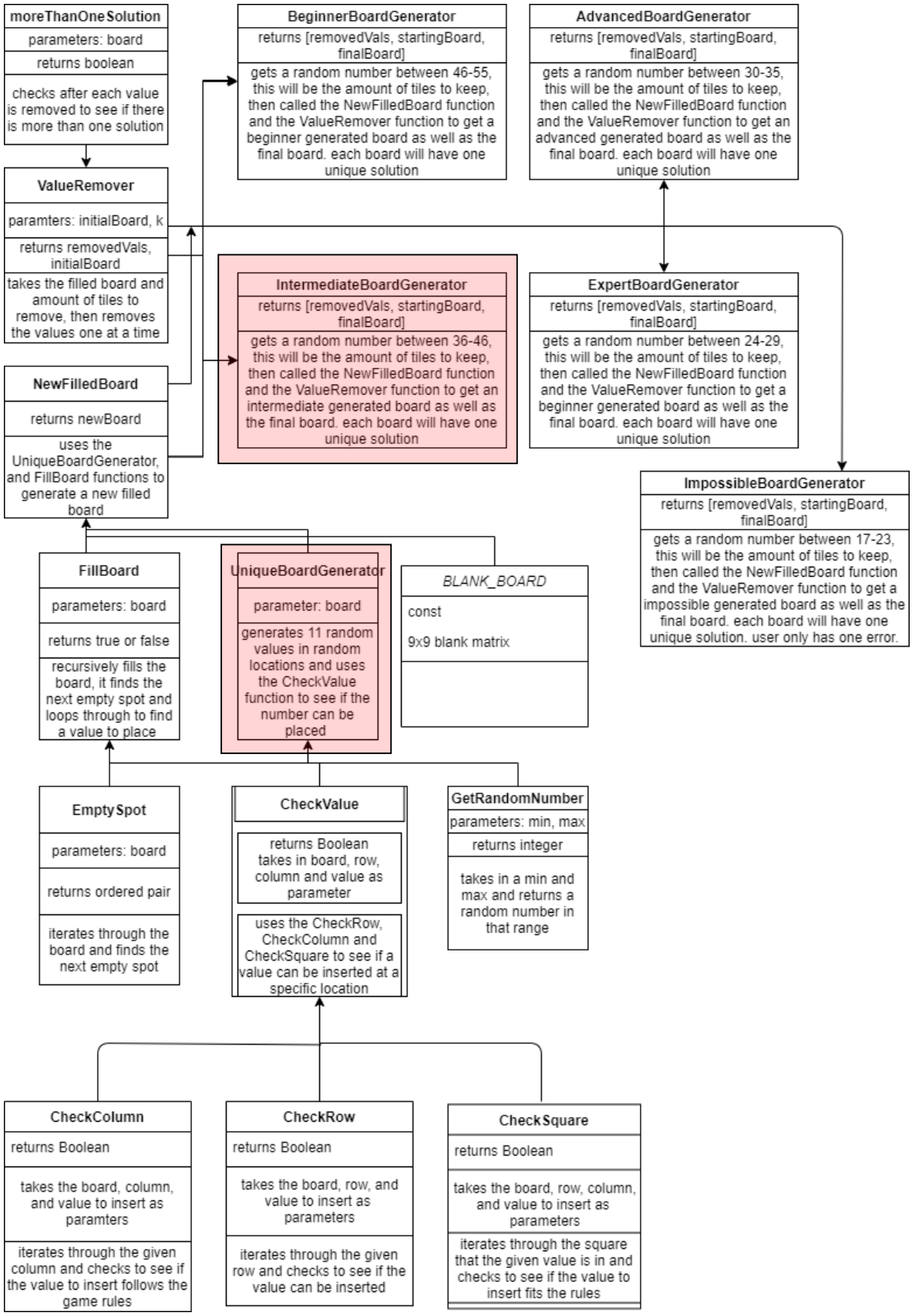
Product Design

Lamb Chops
Daniel Clement, Anna Fortenberry, Spencer Smith, and Sarah Wainwright

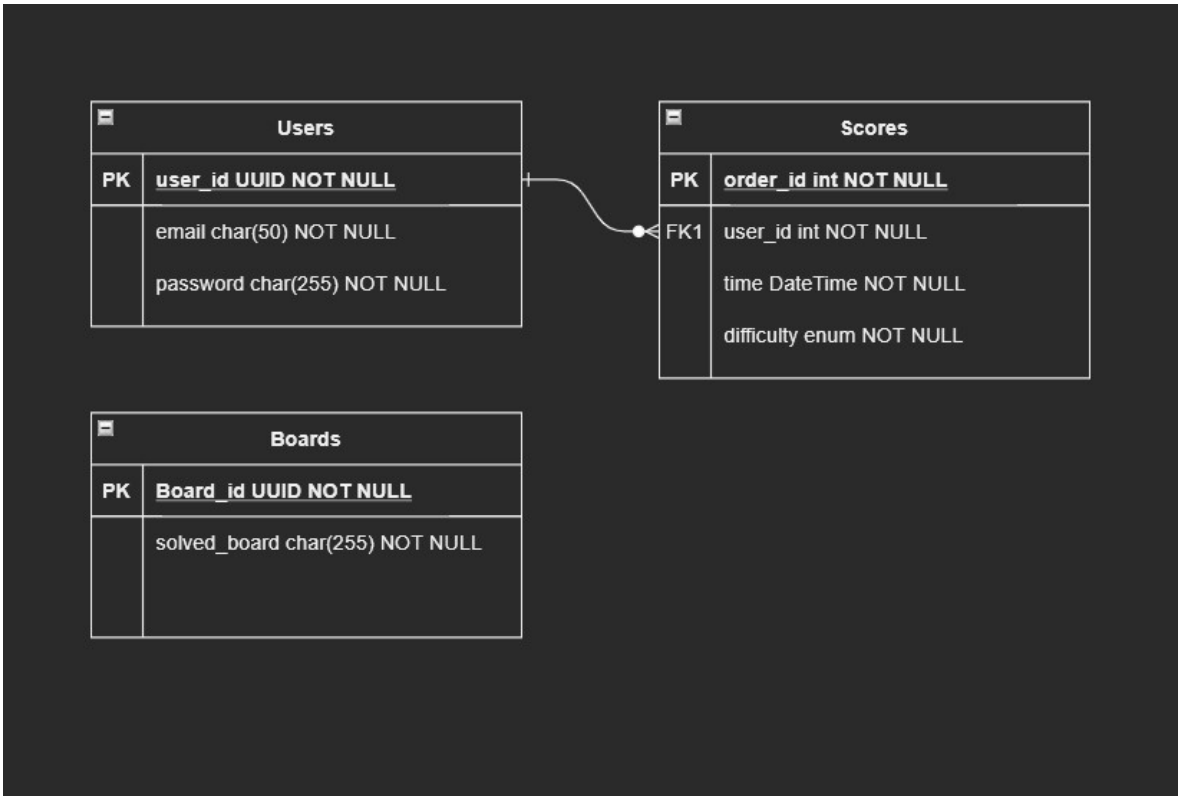
Revision Number	*Revision Date*	*Summary of Changes*	*Author(s)*
0.1	*06/27/21*	*Original*	*Lamb Chops*
0.2	***07/02/21***	***New design for completed game modal.***	***Lamb Chops***
0.3	***07/09/21***	***New logic design for backend – no unique board generator function. Shuffle number array 1-9 instead.***	***Lamb Chops***
0.4	***07/16/21***	***Removed "Impossible" difficulty.***	***Lamb Chops***
0.5	***07/23/21***	***Removed functionality related to a database: save game, personal stats, etc.***	***Lamb Chops***
0.6	***08/01/21***	***Notes mode design finalized and updated.***	***Lamb Chops***

Deletions highlighted in red

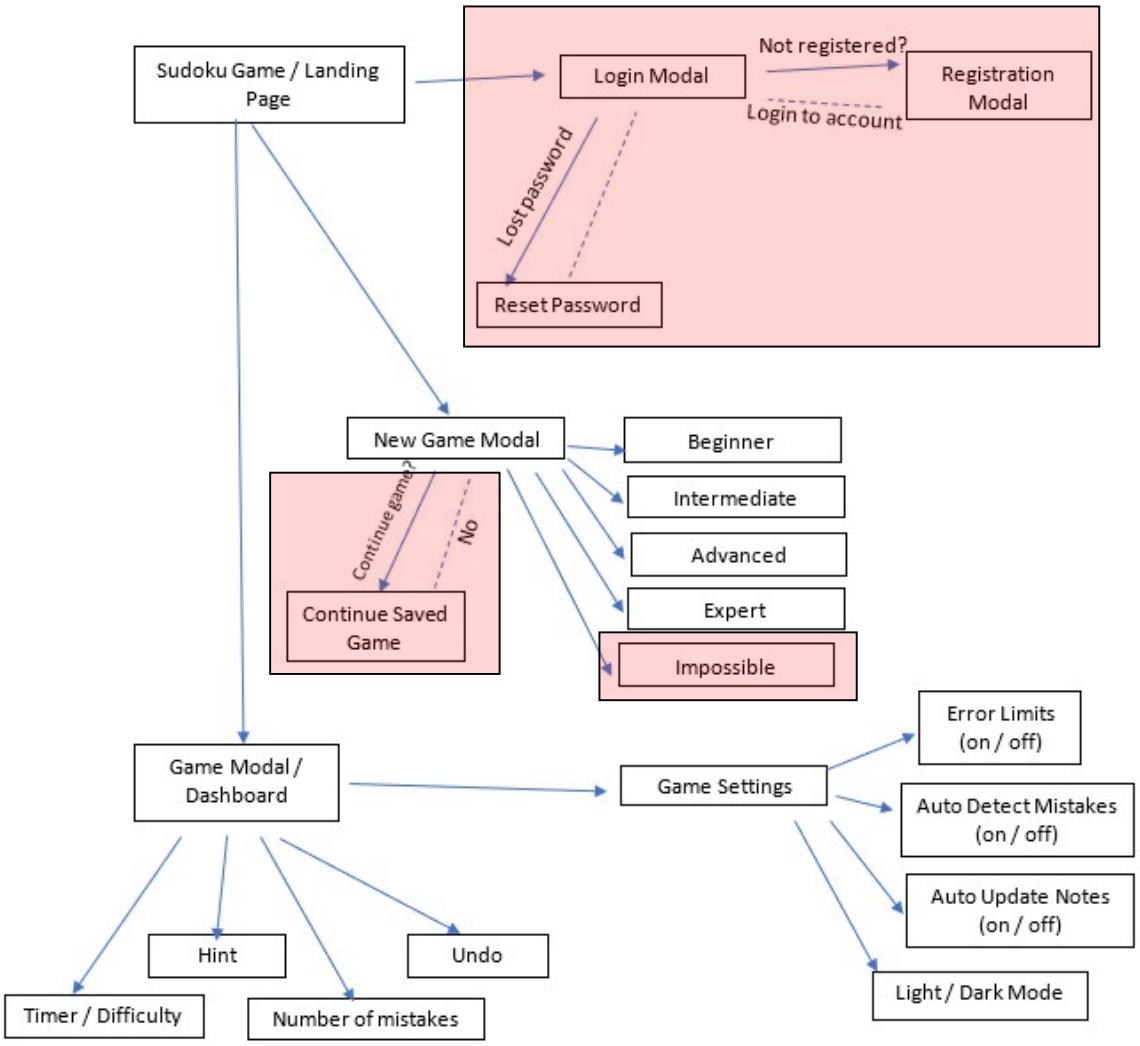
Function Diagram (Our equivalent of "Class Diagram," as relevant to our project)



ER Diagram

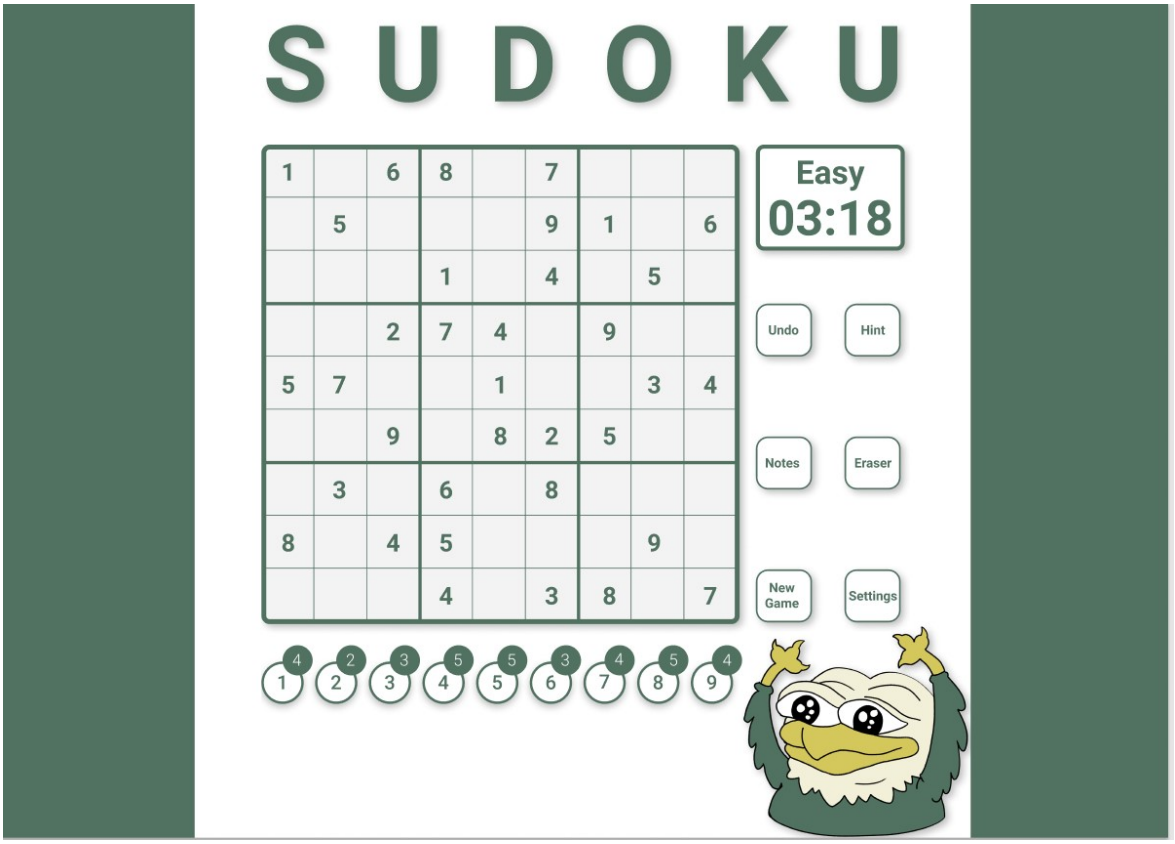

Users
PK
user_id UUID NOT NULL
email char(50) NOT NULL
password char(255) NOT NULL
Scores
PK
order_id int NOT NULL
FK1
user_id int NOT NULL
time DateTime NOT NULL
difficulty enum NOT NULL
Boards
PK
Board_id UUID NOT NULL
solved_board char(255) NOT NULL

Information Architecture Diagram



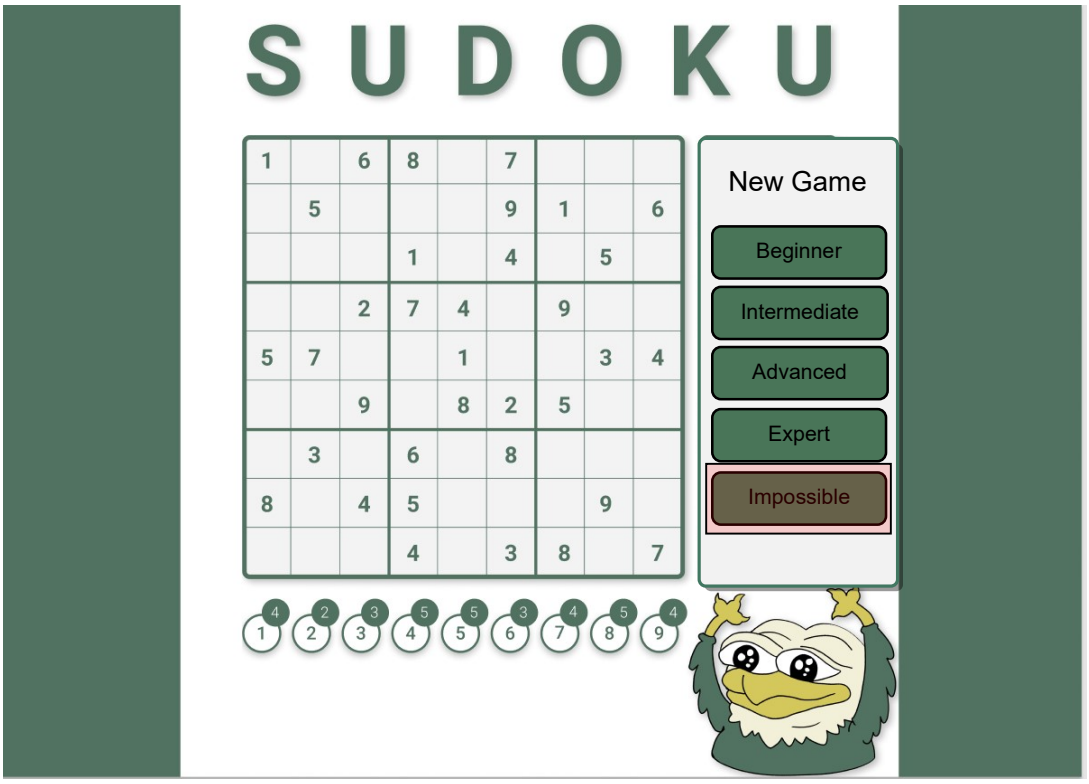
User Interface Wireframe(s)/Screenshot(s)

Landing Page:

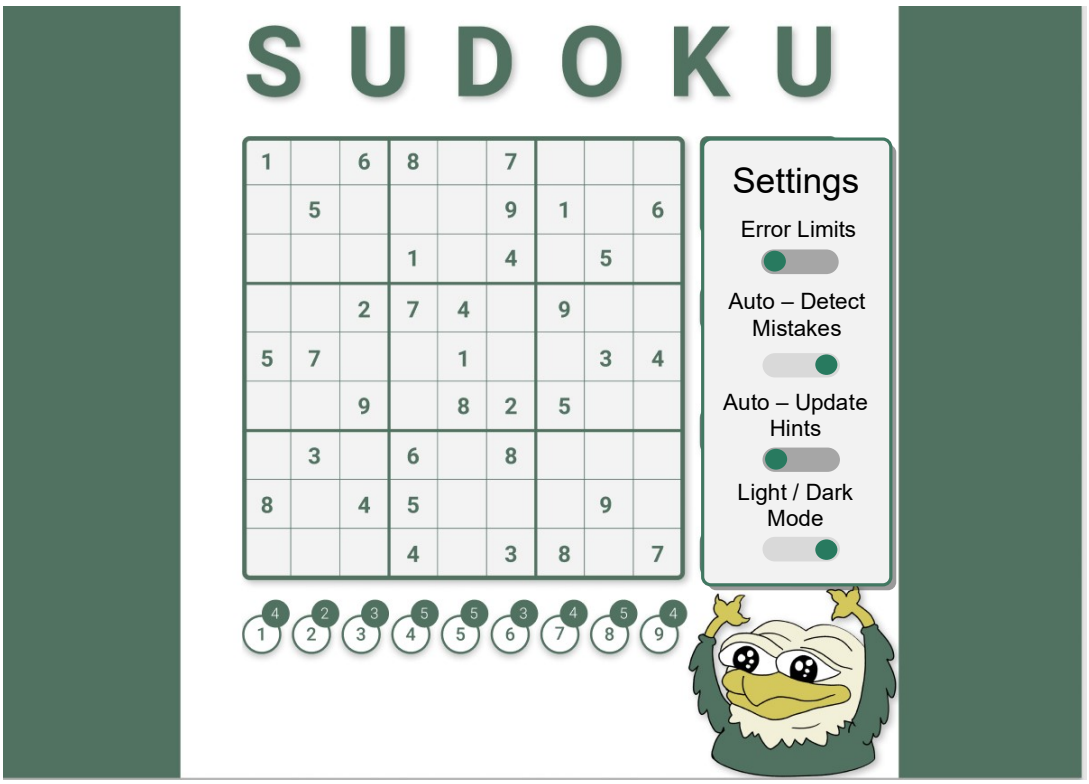


Landing Page Modal (Sprint 6) removed

New Game Modal:



Settings Modal:



Create Account / Login Buttons Modal: removed

Sign-In Modal (Create Account adds “confirm password”): removed

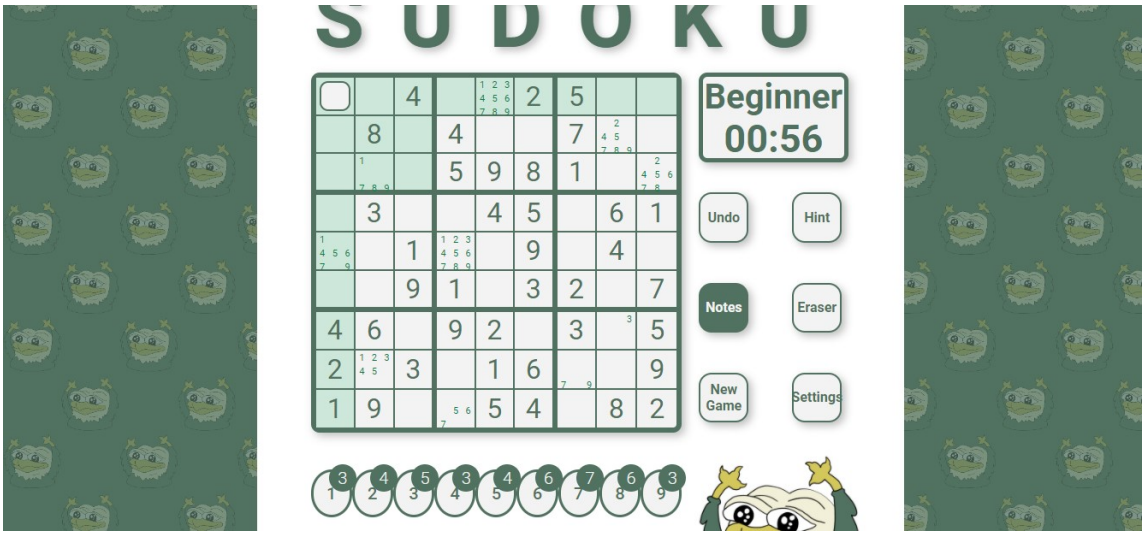
Account Modal: removed

Save Game Overwrite Confirmation: removed

Personal Bests Modal: removed

Completed Game Modal: updated

Notes Mode: updated to this document



Failure Modal:

Design Summary

The front-end of the project will be built using a mix of HTML, CSS, and JavaScript. To make those all work together we are using a JavaScript framework called ReactJs. We are hosting our React application for free on GitHub through a tool called GitHub-pages. The web design was made with the intention of having the user play the game with the start of the application. In other words, there will not be a login required before playing or choosing a difficulty. When the user first visits the site, the program will generate a “Beginner” level Sudoku board, load it to the screen, and start a timer. If the user wants to play on a harder difficulty, they then can click new game and choose it there.

Design Rationale

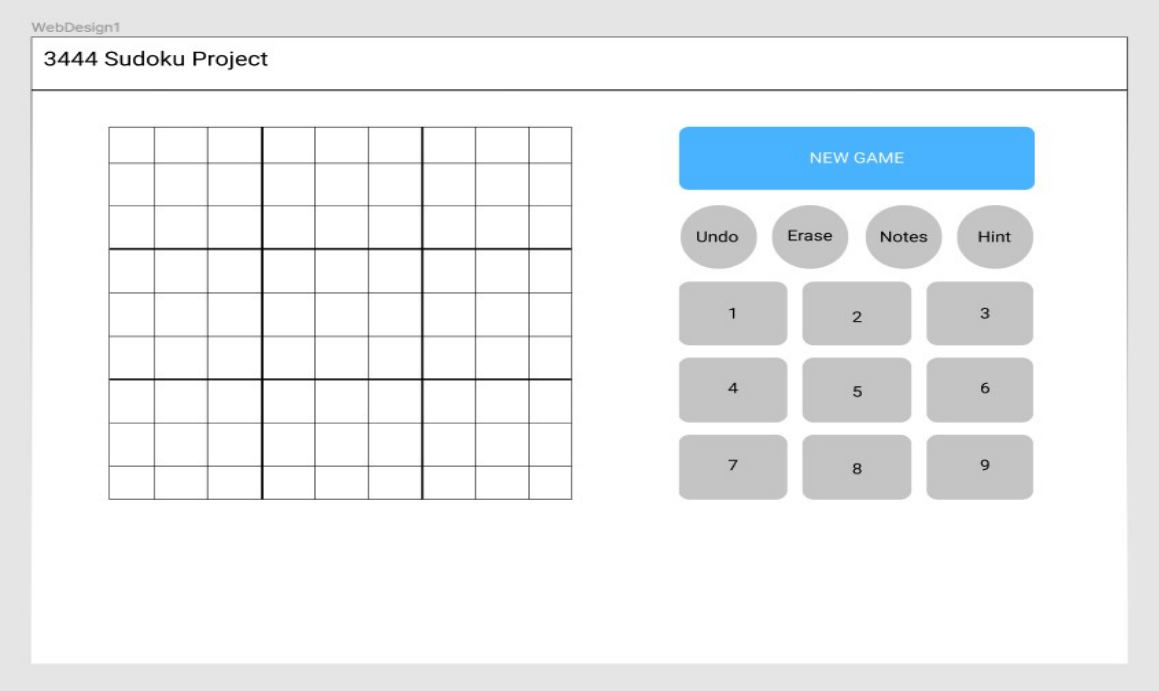
The big decision factors for the project include:

1. Cost
2. Language Barrier
3. Project Scope

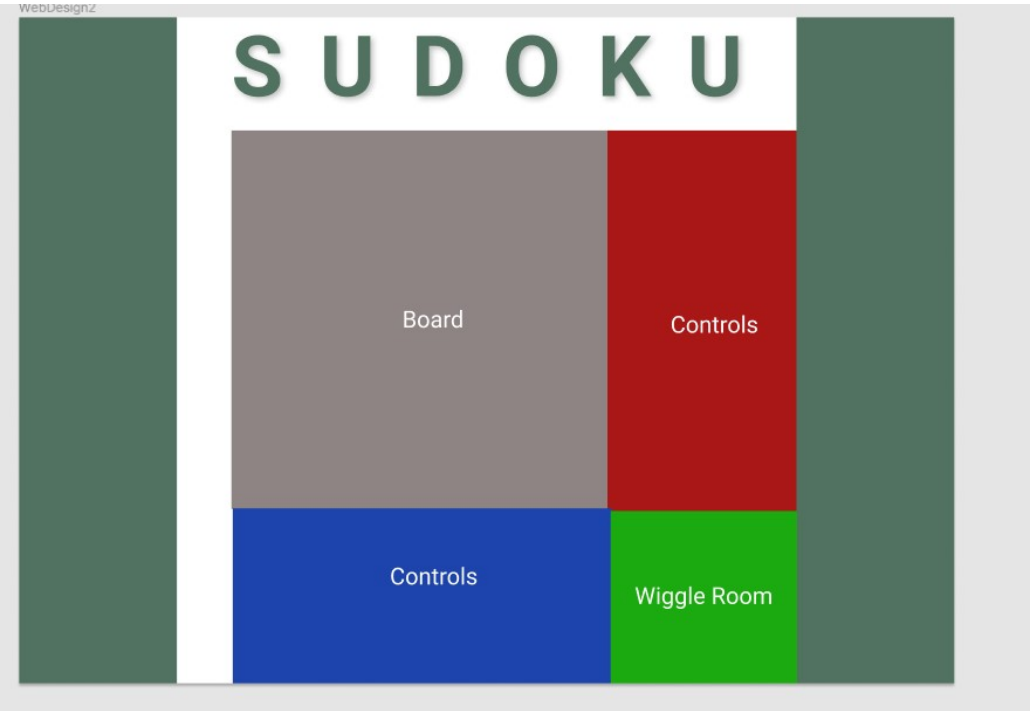
For hosting, we knew from previous web application projects that GitHub has a feature called GitHub-pages. It allows users to host static web pages directly on GitHub for free, so we chose that option as it does everything we need for the project. Additionally, with being free unlike most other options (AWS/AZURE/DigitalOcean)

For Languages we chose to do all the logic of the game in javascript that way it can run in browser along with the front-end framework that handles the html and css for the application. This way we can avoid having to pay for a server to handle the game interactions so everything for this project would be free. Even though most of the group came into this class with experience in C/C++ only, there is no simple way that would not be more trouble than it is worth to use C/C++ in browser. The pros of using javascript ended up out-weighing the cons of learning a whole new language for the project in the summer.

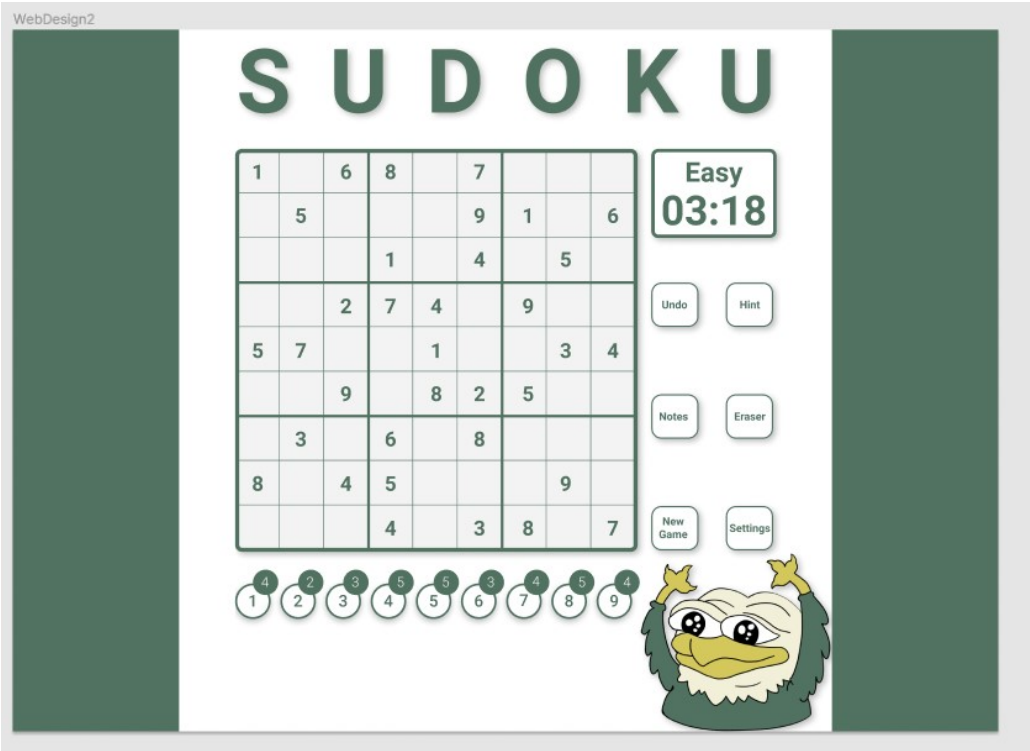
For Web-app design we decided on a single page application to contain everything we would implement. We first decided on all the things that needed to fit on one screen which ended up being a header, the game board, a button for each number, a label that displays the number of tiles remaining for each number, a timer, an undo button, a hint button, a button to switch to a note's mode, an eraser button, a new game button, and a settings button. With that in mind my first iteration of the design was to just put all of that on the screen and this is what I initially came up with.



I did not have to finish this design to realize it was not going to fit the vision I had for it. For one it would not convert to moblile layout very well. Also I did not like the unintended optical illusion effect that the numbers made with the rounded corners. Overall it is not very user friendly and does not have a nice appealing factor in my opinion. So I went back to the drawing board and started from scratch. I then tried to just make block where everything in can fit into the design and started off with this as a proof of concept for the second iteration of the design.

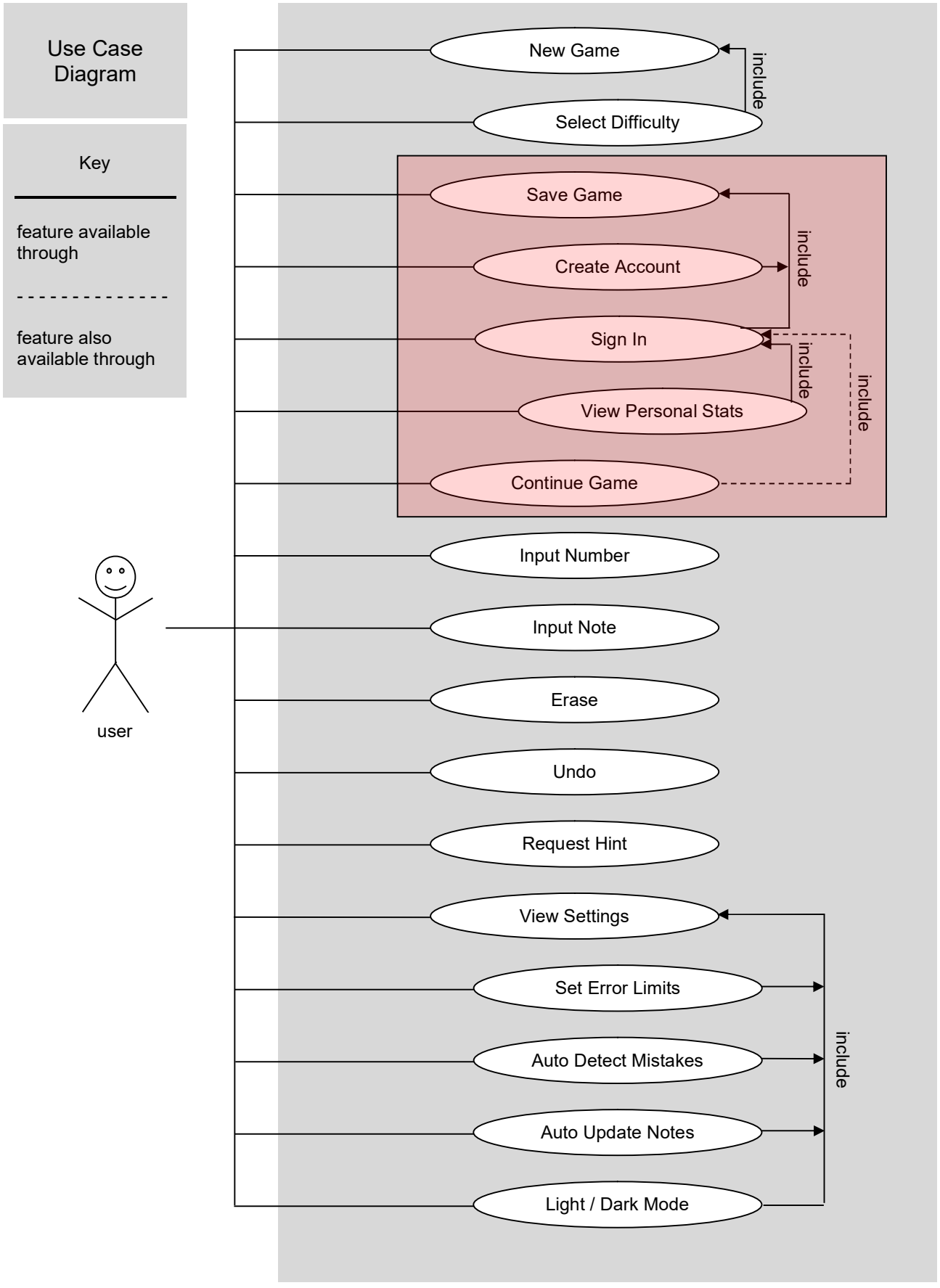


This already was an improvement over the first attempt at laying everything out on the screen. With this layout it would be easier to implement by it all being blocked out in a grid. With this layout for when converting to a mobile responsive layout the main change that must take place is moving the red block's controls below the blue block's controls to make a single column. Happy with this much I moved forward to fill in the blocks with the elements that would occupy them. To make the final iteration of the design before implementation to look like this.



The number buttons and remaining labels I felt just perfectly fit right below the board because it is already split into 9 columns above it and just to continue down and have a button below each of the columns just worked in my head. The right buttons I broke into 2 columns and spaced them out to end up being the length of the board because otherwise there was an awkward bit of white space in the bottom right that I wanted to avoid. That white space being there initially got me thinking of what could go there and it turned into the image that Sarah Wainwright drew herself and I loved it enough to keep it there.

With all of this laid out on the screen it allows for every interaction that the user could do to play sudoku. The user can input the tile by either drag and dropping from the buttons at the bottom or click on a tile then the number. They can make notes by toggling the notes button which will then make all tile inputs not official but just there to take notes on the board. The user can undo a mistake or get a hint or erase all the notes or the number they assigned to a tile. The user can start a new game by clicking the new game button and that will then bring up a pop-up to ask what difficulty. The user can also change the settings.


Use Case Diagram
Key
feature available through
feature also available through
New Game
Select Difficulty
include
Save Game
Create Account
Sign In
View Personal Stats
Continue Game
include
include
include
Input Number
Input Note
Erase
Undo
Request Hint
View Settings
Set Error Limits
Auto Detect Mistakes
Auto Update Notes
Light / Dark Mode
include
user

Use Case ID:	UC - 01
Use Case Name:	Start a new game and generate a board at a specified difficulty level.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User will be able to start a new game and generate a new game board chosen from the following difficulties: “Beginner,” “Intermediate,” “Advanced,” “Expert,” or “Impossible.” The starting number of clues coordinates with these difficulties.
Actor:	User
Pre-conditions:	- System has been set up and configured - System is running - User has accessed website via URL
Scenario Flow:	**Main (Success) Flow:** 1. User selects “New Game” Button beside Sudoku game board. *User may also access this feature with sprint 6 update that adds a modal on the landing screen with buttons “New Game” and “Continue Game.”* 2. A new modal pops up with buttons labeled with the following difficulties: “Beginner,” “Intermediate,” “Advanced,” “Expert,” and “Impossible.” 3. A new Sudoku board is generated with the correct number of starting hints (for Impossible, a 16x16 game board will appear.
Alternate Flows:	**UC - 01.1** The top right corner of the New Game modal, which displays the different levels of difficulty, there is an “x.” 1. User clicks on the “x.” 2. The modal exits and the current Sudoku board is uncovered. If it is an unfinished game, the user may continue playing.
Post Condition:	In the success flow, a new game board of correct difficulty is generated. The new game board is saved to memory and all saved states of the previous board are cleared from memory.

	In the alternate flow, the board currently initialized is uncovered, whether it is complete or not.

Use Case ID:	UC - 02
Use Case Name:	Place a playable number *or letter* on the game board.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User will be able to choose and place a playable number or letter for the current game board on a square.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized
Scenario Flow:	**Main (Success) Flow:** 1. User selects the box they wish to fill. 2. User selects the desired number or letter from the buttons located beneath the game board. For difficulties “Beginner” through “Expert” the buttons include numbers 1 – 9. For “Impossible,” the letters A – G are available in addition to the numbers. 3. The selected number or letter displays in the box. If the box was previously filled by the user, the previously stored number will be overwritten. 4. Repeats steps 1 – 3 as many times as desired.
Alternate Flows:	*User will not be able to select a box that was initialized with the generation of the board, so there is no alternative flow for that error.*
Post Condition:	User has placed an intended number or letter in a box of their choice. The system saves the new state of the board to memory.

Use Case ID:	UC - 03
Use Case Name:	Remove a number or letter from the game board.

Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User will be able to remove a number they played previously.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized - User has played numbers or letters on the game board (UC - 02)
Scenario Flow:	**Main (Success) Flow:** 1. User selects the box they wish to remove a number or letter from. 2. User selects the coordinating number or letter from the buttons located beneath the game board. 3. The selected number or letter displays in the box. If the box was previously filled by the user, the previously stored number will be overwritten.
Alternate Flows:	*User will not be able to select a box that was initialized with the generation of the board, so there is no alternative flow for that error.* **UC – 03.1** 1. User selects a number or letter that does not coordinate with the number or letter displayed in their selected box. 2. The system overwrites the currently displayed number or letter with the number or letter clicked on by the user.
Post Condition:	In the success flow, the user has removed their desired number or letter. In the alternate flow, the user has overwritten their desired number or letter with an undesired number or letter. In either flow, the system saves the new state of the board to memory.

Use Case ID:	UC - 04
Use Case Name:	User makes a note on the game board.
Created By:	Lamb Chops

Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User will be able to place a note of a playable number or letter for the current game board on a square.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized
Scenario Flow:	**Main (Success) Flow:** 1. User selects the button labeled “Note.” 2. User selects the box they wish to make a note on. 3. User selects the desired number or letter from the buttons located beneath the game board. 4. The selected number or letter displays smally in the box. If the box was previously filled by the user in game mode, the number will be overwritten. If the box contains other, different notes, the new note will display in addition to the old one. 5. Repeats steps 2 – 4 as many times as desired.
Alternate Flows:	*User will not be able to select a box that was initialized with the generation of the board, so there is no alternative flow for that error.*
Post Condition:	User has made a note of an intended number or letter in a box of their choice. The system saves the new state of the board to memory.

Use Case ID:	UC - 05
Use Case Name:	User removes a note from the game board.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User will be able to remove a note they played previously.

	The system saves the new state of the board to memory.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized - User has made a note on the game board (UC - 04)
Scenario Flow:	**Main (Success) Flow:** 1. User selects the box they wish to remove a number or letter from. 2. User selects the coordinating number or letter from the buttons located beneath the game board. 3. The selected number or letter displays in the box. If the box was previously filled by the user, the previously stored number will be overwritten.
Alternate Flows:	*User will not be able to select a box that was initialized with the generation of the board, so there is no alternative flow for that error.* **UC – 05.1** 1. User selects a number or letter that does not coordinate with the number or letter displayed in their selected box. 2. The system overwrites the currently displayed number or letter with the number or letter clicked on by the user.
Post Condition:	In the success flow, the user has removed their desired number or letter. In the alternate flow, the user has overwritten their desired number or letter with an undesired number or letter. In either flow, the system saves the new state of the board to memory.

Use Case ID:	UC – 06
Use Case Name:	Clear a game box on the game board (erase).
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User can clear a square of any number or notes displayed.

Actor:	User
Pre-conditions:	- A game board has been successfully initialized - User has made a play on the game board (UC – 02 / UC – 04)
Scenario Flow:	**Main (Success) Flow:** 1. User selects the square on the board that they wish to clear. 2. User selects the “Erase” button. 3. The selected box is cleared.
Alternate Flows:	*There is no “failure” to this flow because if the user changes their mind, they can undo the change (UC – 07).*
Post Condition:	The system has cleared the square selected and cleared by the user of all data, regardless of whether it displayed a game board play or note(s). The system saves the new state of the board to memory.

Use Case ID:	UC – 07
Use Case Name:	Undo a change made to the game board.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user can undo a play made on the board (display the previous state of the board).
Actor:	User
Pre-conditions:	- A game board has been successfully initialized - User has made a play on the game board (UC – 02 / UC – 04)
Scenario Flow:	**Main (Success) Flow:** 1. The user selects the button “Undo” located beside the game board. 2. The last change made to the board by the user is undo, regardless of

	whether it was a placement of a number or the removal of a number(s).
Alternate Flows:	n/a
Post Condition:	The system calls for and displays the previous state of the board (pops the newest saved state).

Use Case ID:	UC – 08
Use Case Name:	Request a hint.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user may request a hint. A hint randomly fills an unfilled square with the correct answer. *Consider: Should hint function also be able to correct a wrong answer? What if it causes multiple violations at once (when error limits are turned on)? How are errors detected when auto-detect mistakes is off?* *Update: Hint corrects wrong answers and fills blank cells. The first instance of an error is correct first, and the first instance of a blank cell is corrected if there are no errors. There is a limit of three hints for any difficulty.*
Actor:	User
Pre-conditions:	- A game board has been successfully initialized
Scenario Flow:	**Main (Success) Flow:** 1. User requests a hint by selecting the “Hint” button beside the board. 2. A random unfilled box is detected and filled with the correct answer. 3. The hint count is reduced by one.
Alternate Flows:	**UC – 08.1** 1. User requests a hint when their remaining hint count equals zero. 2. An error message is displayed, and a new hint is not displayed.

Post Condition:	In the success flow, the user has received a new hint in a random location. The system saves the new state of the board to memory. In the alternative flow, the system does not authorize a new hint.

Use Case ID:	UC – 09
Use Case Name:	Change the game settings.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user displays the settings menu and makes changes to one or multiple of the available options.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized
Scenario Flow:	**Main (Success) Flow:** 1. User clicks on the settings button. 2. A modal displaying options for “Error Limits,” “Auto-Detect Mistakes,” “Auto-Update Notes,” and “Light/Dark Mode” pops up on the screen. a. If the game mode is set to a difficulty of “Impossible,” the “Error Limits” feature is disabled. 3. User may turn on or off any of the settings.
Alternate Flows:	**UC – 09.1** 1. User decides they want to close the settings and clicks the “X” in the top right corner of the modal. 2. The modal closes.
Post Condition:	In the success flow, an altercation has been made to the game settings and saved in the system. In the alternate flow, the user exits before any changes are made, and the system remains as it was before.

Use Case ID:	UC – 10
Use Case Name:	Turn off restriction on errors allowed per game.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user changes the setting that restricts the number of errors a user can make in a game. For “Beginner,” the number of errors allowed is 10, for “Intermediate,” 5, for “Advanced,” 3, and for “Expert,” 1. This feature is disabled in “Impossible” mode.
Actor:	User
Pre-conditions:	- User has successfully displayed the game settings.
Scenario Flow:	**Main (Success) Flow:** 1. User flips the switch to turn off the setting enabling error restrictions. 2. User may now make as many mistakes as they want without the game terminating.
Alternate Flows:	**UC – 10.1** 1. User decides they want to keep error limits on. They flip the switch to turn them back on. 2. User can make a limited number of mistakes before the game board tells them they failed.
Post Condition:	In the success case, the system settings no longer check the error count after each mistake made by the user. In the alternative case, the system saves the changes to the system twice. The system is set up to run as it did before the user made a change to the settings.

Use Case ID:	UC – 11
Use Case Name:	Turn on setting to auto-detect mistakes.
Created By:	Lamb Chops
Last Updated By:	n/a

Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user enables the setting that allows the system to detect when they place a number that conflicts with the answer key. With the setting off, users are only notified when they make a play that breaks the rules of Sudoku (meaning they can make several wrong plays without realizing it).
Actor:	User
Pre-conditions:	- User has successfully displayed the game settings.
Scenario Flow:	**Main (Success) Flow:** 1. User flips the switch to turn on the setting enabling auto-detection of mistakes. 2. User is notified when they play a number that does not match the correct solution of the game board. 3. Error count is incremented for every mistake.
Alternate Flows:	**UC – 11.1** 1. User decides they want to keep auto-detection of mistakes off. They flip the switch to turn them back off 2. User will only be notified of a mistake if they violate a rule of sudoku. Error count will only increment when this occurs.
Post Condition:	In the success case, the system confers with the game solution after every play made by the user. The number of allowed errors is decremented by one every time the user makes a play that does not match it, and the user is notified of this error. In the alternative case, the system saves the changes to the system twice. The system is set up to run as it did before the user made a change to the settings. This means, the system does not call on the back-end code for the stored game key. It checks whether each play follows the rule of Sudoku via the front-end code. The error count is only decremented when the user violates one of the rules.

Use Case ID:	UC – 12
Use Case Name:	Turn on setting to auto-update notes with placement of number.
Created By:	Lamb Chops
Last Updated By:	n/a

Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	All notes that violate the rules of Sudoku when a play is made on the board are automatically removed. This means, if the user plays a 1, all noted 1's in the coordinating row, column, and box will no longer display.
Actor:	User
Pre-conditions:	- User has successfully displayed the game settings.
Scenario Flow:	**Main (Success) Flow:** 1. User flips switch for "Auto-Update Notes." 2. Notes that violate breaks of Sudoku when user makes a play will automatically update.
Alternate Flows:	**UC - 12.1** 1. User decides they do not want this feature enabled and flips switch back off. 2. System settings that detect notes that violate the law of Sudoku are turned off.
Post Condition:	In the success flow, the system settings are changed to check whether any of the notes in the coinciding row, column, and square match a number/letter played by the user. If a match is found, the system no longer displays the note. The new state of the board is saved on the front-end because notes are independent of the backend. In the case of the alternate flow, the settings above are saved first. Then, the settings are reverted to what they were before the user flipped the switch. This means the user must manually check the notes they made and see if any conflict.

Use Case ID:	UC – 13
Use Case Name:	Activate dark mode application display.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a

Overview:	The user may adjust the display settings to dark mode. This alters the colors to be easier on the eyes in dark settings.
Actor:	User
Pre-conditions:	- User has successfully displayed the game settings.
Scenario Flow:	**Main (Success) Flow:** 1. User flips the switch for “Light/Dark Mode.” 2. All visible features of the web application adjust for dark mode.
Alternate Flows:	**UC – 13.1** 1. User decides they do not want this feature enabled and flips switch back off. 2. System reverts colors back to light mode.
Post Condition:	In the success flow, the system adjusts the display colors to compensate for a darker environment. In the alternate flow, the system first saves the settings for dark mode. It then switches all display features back to light mode and saves them again.

Use Case ID:	UC – 14 (Removed)
Use Case Name:	Save the current game board.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user can save their current game board in memory to their account.
Actor:	User
Pre-conditions:	- A game board has been successfully initialized
Scenario Flow:	**Main (Success) Flow:** 1. User selects the “Save” button located beside the Sudoku game

	board. 2. A modal is displayed with the buttons “Create Account” and “Sign In.” *If the user signed in at the beginning of the game with the “Continue Game” feature, they will bypass the remaining steps, and the system will automatically save their game. User will be notified that their game was successfully saved and exit the modal.* 3. User chooses to create account (UC – 15) and successfully registers. a. User chooses to sign in (UC – 16) and is successfully authenticated. 4. System saves the game in memory attached to the user’s account login information. a. If the user has a game saved already, the user is prompted with a confirmation to overwrite a preexisting game. User confirms or denies.
Alternate Flows:	**UC – 14.1** 1. Users exits the modal without successfully signing in. 2. Modal exits and game board displays as it was before, but it is not backed up. **UC – 14.2** 1. Web application is shut down in the middle of a game. 2. If user reopens the application, a new, unrelated game board will be generated. Their previous game board is not saved.
Post Condition:	In the successful flow, the system recognizes the current user’s account (they are logged in), and their requested game board has been saved to memory in its present state (this means if the user makes updates afterwards and does not save the board again, the updates will not be stored). *If the user has a game board saved to their account already, UC – 20 will execute.* a. User’s game board overwrites a game board or denies confirmation and UC -14.1 executes. In the first alternate flow, the system does not save the existing game board. The modal exits and the board is uncovered as it was before. In the second alternate flow, the game board does not save, and the user loses their current progress. When the application is reopened, the landing page modal is redisplayed.

Use Case ID:	UC – 15 (Removed)
Use Case Name:	Create Account
Created By:	Lamb Chops

Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user is able to create an account for the purpose of saving one game board and tracking their personal best times.
Actor:	User
Pre-conditions:	- User has opted to “Save Game.”
Scenario Flow:	**Main (Success) Flow:** 1. User does not have an existing account and selects “Create Account.” 2. The registration portal opens, which displays blanks for “username,” “password,” and “confirm password.” 3. The user fills in the blanks with the corresponding information. *If the user’s password blanks do not match, the system will not allow them to proceed.* 4. The user’s account is created, and they are automatically logged in. 5. Their game board is saved.
Alternate Flows:	**UC – 15.1** 1. The user exits the registration modal. 2. The game board is uncovered, and the user remains anonymous to the system.
Post Condition:	In the success flow, the system saves the current game board to an account with the user’s specified username. If the user opts to save their game from this point on, their game board saved will automatically update, for the duration of their session on the web application (uninterrupted). In the alternate flow, the user remains anonymous to the system, and nothing will be saved outside of volatile memory.

Use Case ID:	UC – 16 (Removed)
Use Case Name:	Sign - In
Created By:	Lamb Chops
Last Updated By:	n/a

Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user may sign-in to their preexisting account.
Actor:	User
Pre-conditions:	- User has either opted to “Continue Game” on the landing screen modal, or the user has opted to “Save Game” by clicking the button next to the game board.
Scenario Flow:	**Main (Success) Flow:** 1. A modal pops up with blanks for username and password. 2. The user inputs information that matches their account. 3. The user is now logged into the system and can view their saved data. a. UC - 17 b. If the user accessed this feature through “Save Game,” their game will be saved in secondary memory.
Alternate Flows:	**UC – 16.1** 1. The user input incorrect account information. 2. They are informed that the username does not exist or their password in incorrect. 3. The sign-in screen redisplays blank. **UC – 16.2** 1. The user exits the sign-in modal.
Post Condition:	For the success flow, the user is recognized by the system and has access to all features that correlate. They may now save their game without needing to sign-in again. For the first alternate flow, the user continues to attempt to sign in until they succeed (UC – 16) or exit (UC – 16.2). For the second alternate flow: a. UC 17.1 b. If the user previously selected to “Save Game,” their progress will not be saved to secondary.

Use Case ID:	UC – 17 (Removed)
Use Case Name:	Continue Game
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user may continue a game they previously saved.
Actor:	User
Pre-conditions:	- User has previously created an account. - User has saved a game to their account.
Scenario Flow:	**Main (Success) Flow:** 1. User selects “Continue Game” on the landing page modal. *User may also access this feature from the sign-in modal, flipping steps 1 and 2.* 2. User is prompted with the sign-in modal (UC – 16). 3. The user’s game board saved to their account is displayed.
Alternate Flows:	**UC 17.1** 1. See UC 16.2
Post Condition:	In the success flow, the user’s last saved game board will appear on the screen. In the alternate case, the Sudoku board will display blank. To start a game, they will have to select the “New Game” button.

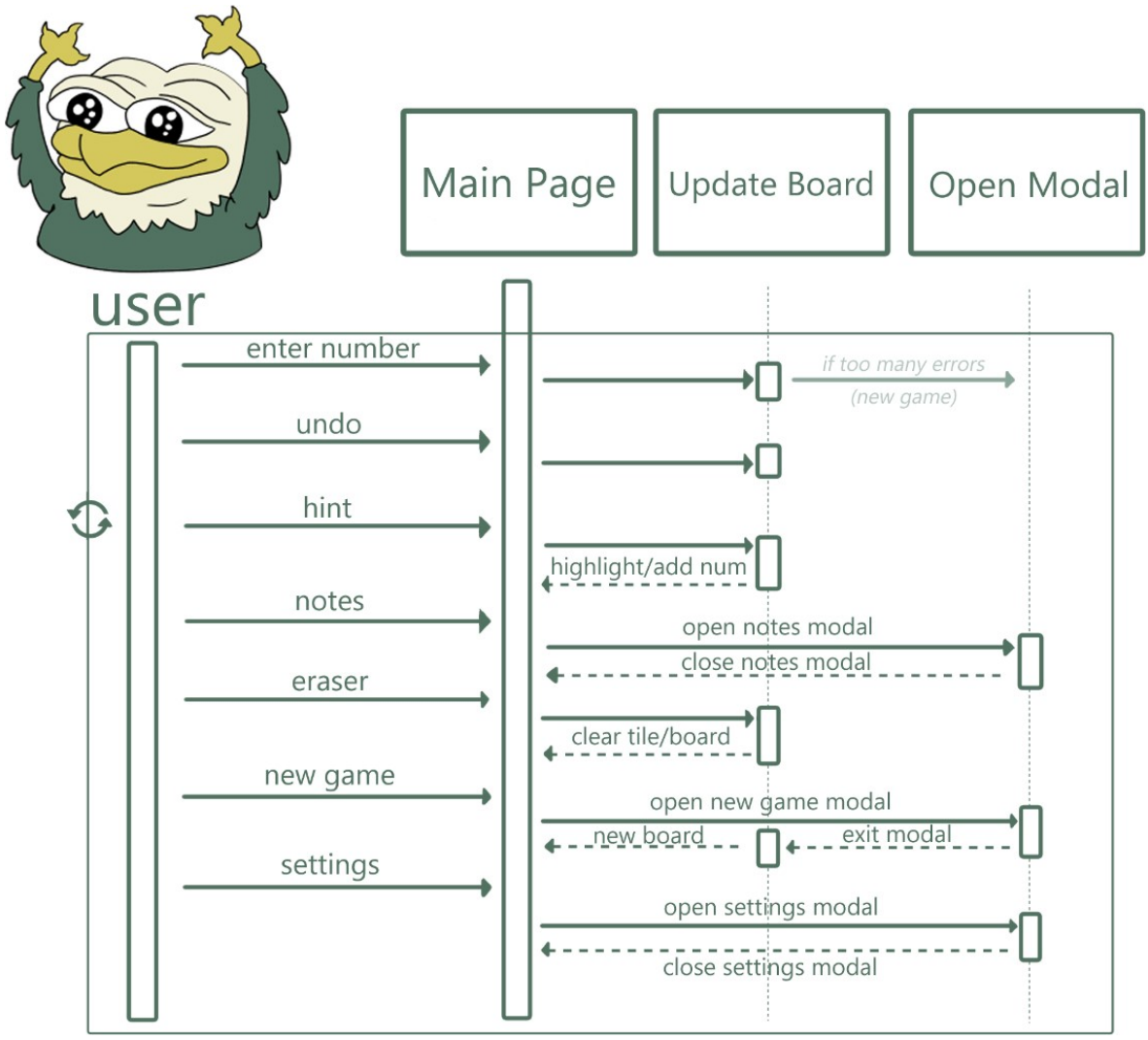
Use Case ID:	UC – 18 (Removed)
Use Case Name:	View Personal Bests
Created By:	Lamb Chops
Last Updated By:	n/a

Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	User may view their best times for each level of difficulty.
Actor:	User
Pre-conditions:	- User has created an account. - User has logged in.
Scenario Flow:	**Main (Success) Flow:** 1. User selects “View Personal Bests” in the modal that appears after signing in. 2. Modal appears displaying the fastest time they have completed a puzzle in each level of difficulty.
Alternate Flows:	n/a
Post Condition:	The system calls for and displays the best times attached to the user’s account information for each level of difficulty. No altercations to the board or system are made.

Use Case ID:	UC – 19
Use Case Name:	User is congratulated for completing a Sudoku game board.
Created By:	Lamb Chops
Last Updated By:	n/a
Date Created:	06/25/21
Date Last Updated:	n/a
Overview:	The user completes a game board successfully and is congratulated.
Actor:	User
Pre-conditions:	- Game board has been successfully initiated and solved.
Scenario Flow:	**Main (Success) Flow:**

	1. User solves Sudoku Puzzle. 2. System saves the time it took to complete the puzzle and displays it on the screen for the user to see. a. If the user is signed in, the time will be compared to the time saved to their account for that difficulty (just stored if no time saved yet). If it is lower, the new time replaces the previous. 3. If the user is signed in and just completed a game through the "Continue Game" option, the memory is freed.
Alternate Flows:	**UC 19.1** 1. The user exceeds the allowed number of errors before they solve the puzzle. 2. They fail and can no longer make changes to the board.
Post Condition:	In either case, the board is as a stand still. They must select "New Game" to play another. In the success flow, if the user is signed in, new data has been updated to their account.

Sequence Diagram(s)



Applicable to the following user cases:

UC – 01(.1)

UC – 02(.1)

UC – 03(.1)

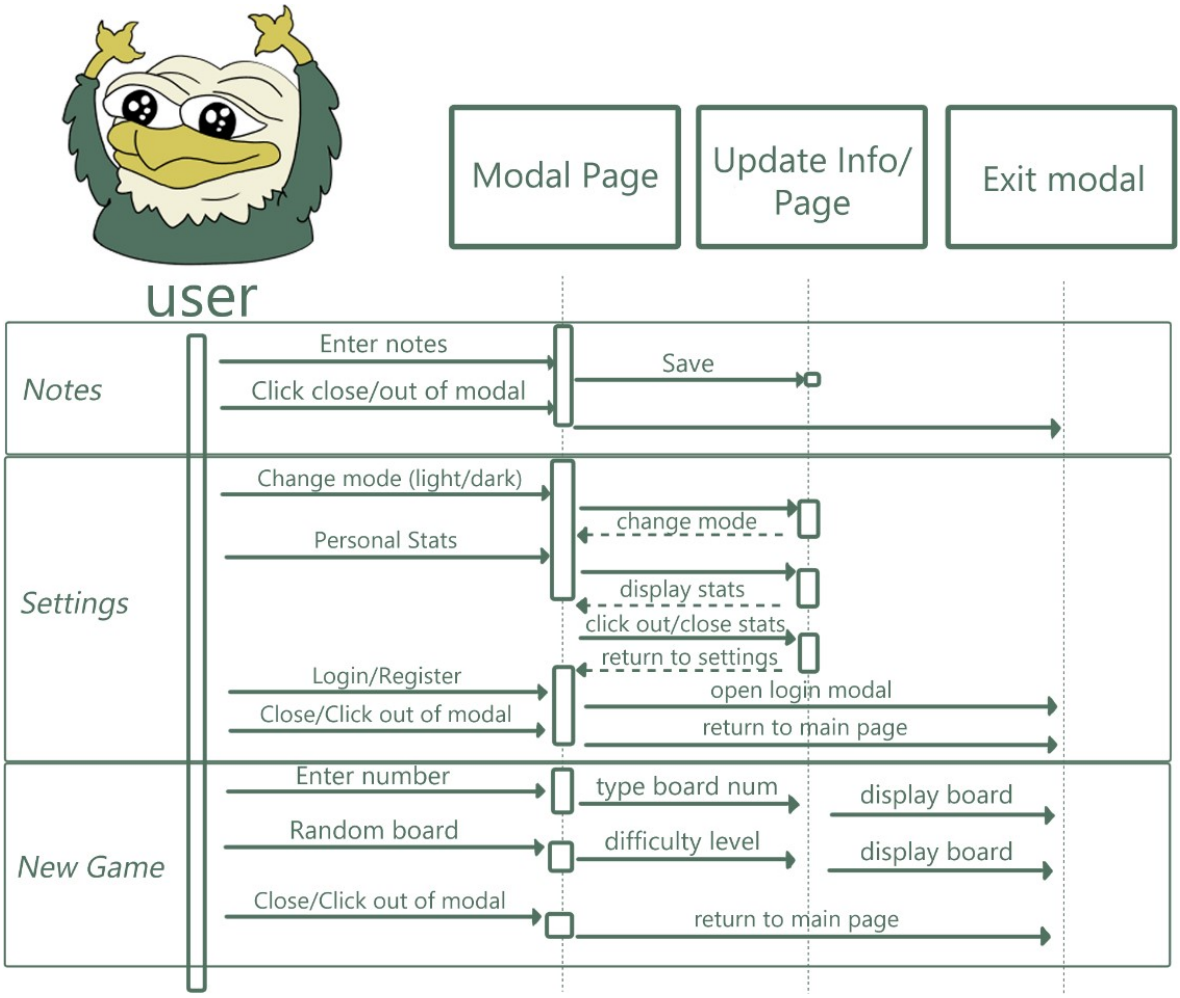
UC – 04/05(.1)

UC – 06

UC – 07

UC – 08(.1)

UC – 09(.1)



Applicable to the follwing user cases:

UC – 01(.1)

UC – 04/05(.1)

UC – 10(.1)

UC – 11(.1)

UC – 12(.1)

UC – 13(.1)