Sudoku

Group 7 Project Design, Rev 14

CMSC 495 (6380) Current Trends and Projects in Computer Science (2205)

University of Maryland Global Campus

Summer 2020 – Professor Hung Dao

July 6, 2020

Group 7

Ben Brandhorst

Christopher Breen

Christopher Smith

Revision	Date	Description	Contributor
1	6/9/2020	Cover page and revision table created; listed seven scenarios; completed description and pre/post conditions for scenarios 7-10.	Chris Breen
2	6/11/2020	Completed scenarios 1-6 (startup, shutdown, and four error-handling scenarios).	Chris Smith
3	6/11/2020	Added pseudocode for the following subsystems: Input, NewGame, CreateGame, Play, UserPageSelection	Chris Breen
4	6/11/2020	Added pseudocode for GameRecords and Leaderboard subsystems	Chris Smith
5	6/11/2020	Event Trace Diagrams added for Scenarios 1, 2, 3, 4	Ben Brandhorst
6	6/12/2020	Event Trace Diagrams added for Scenarios 5, 6, 7, 8, 9, 10, 11	Ben Brandhorst
7	6/12/2020	Class/Function Diagram added, formatting and style adjustments	Chris Breen
8	6/16/2020	Adjusted verbiage of scenario 6 based on Nicholas Groth peer review	Chris Breen
9	6/21/2020	Split ‘Play’ scenario into multiple scenarios to cover smaller blocks of game play	Chris Breen
10	6/23/2020	Updated scenario 3 and 4 descriptions based on Zachary Finnegan feedback.	Chris Breen
11	6/24/2020	Added Scenario 14 for uploading of puzzles	Chris Breen
12	7/1/2020	Added concern for Chrome regarding lack of SameSite attribute in header	Chris Smith
13	7/2/2020	Added Scenario 15 for Hints	Chris Breen
14	7/6/2020	Added Event Trace Diagrams for Scenario 14, 15	Ben Brandhorst

~~CLASS~~ FUNCTION DIAGRAM (Our Python Web App will not be using classes)

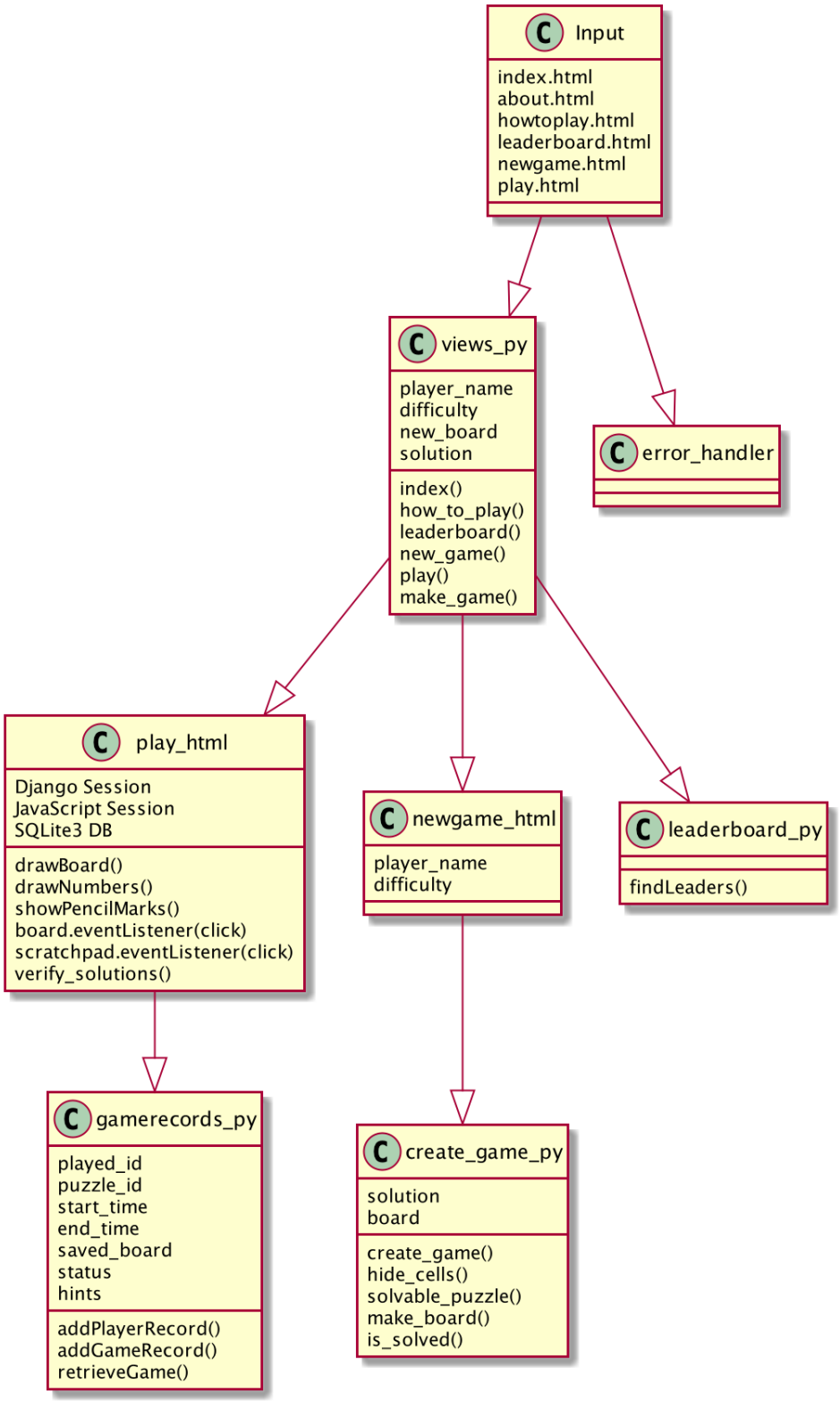

Figure 1 - Class/Function Diagram

EVENT-TRACE (SEQUENCE) DIAGRAMS

Scenario 1: Startup

Description: The web server starts, initializes the database, Django apps, and middleware; begins listening for HTTP connections on port 80.

Pre-condition: AWS Elastic Beanstalk environment created; application packaged and uploaded to AWS.

Post-condition: A user may now interact with the application after connecting to the web server

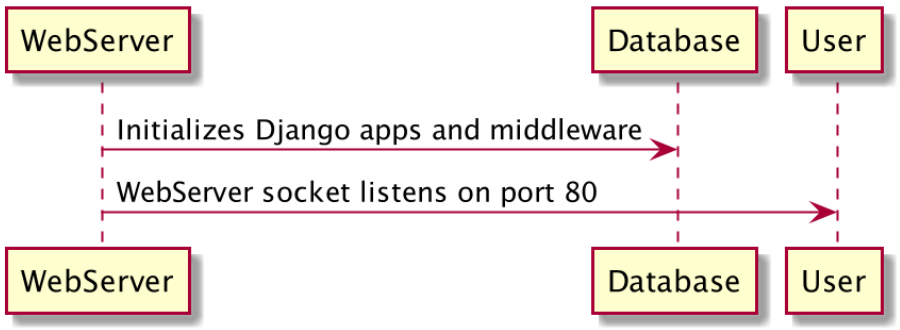

Figure 2 - Scenario 1: Startup

Scenario 2: Shutdown

Description: The web server is expected to run continuously, thus once this course concludes, the webserver closes the database, the web server stops listening for connections, and all systems shut down.

Pre-condition: The web server is running, constantly listening for connections.

Post-condition: The application is shut down, preventing any interactivity until the web server is running again.

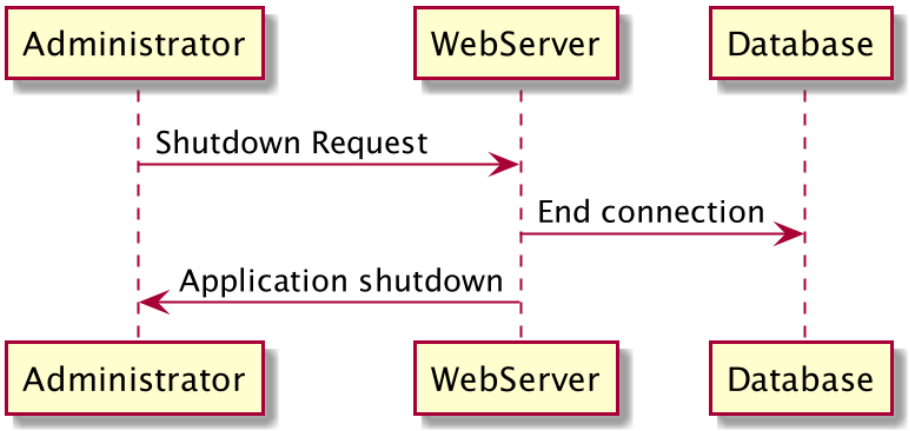

Figure 3 - Scenario 2: Shutdown

Scenario 3: Error-handling—Invalid difficulty entry

Description: POST data and JavaScript session data are vulnerable to user tampering. Invalid difficulty user input prevents board creation in lieu of creating a board with undefined difficulty/techniques.

Pre-condition: User visited the home page, clicked Start a New Game, and newgame.html has been rendered.

Post-condition: A board is not created and the user is given the option to create a board again.

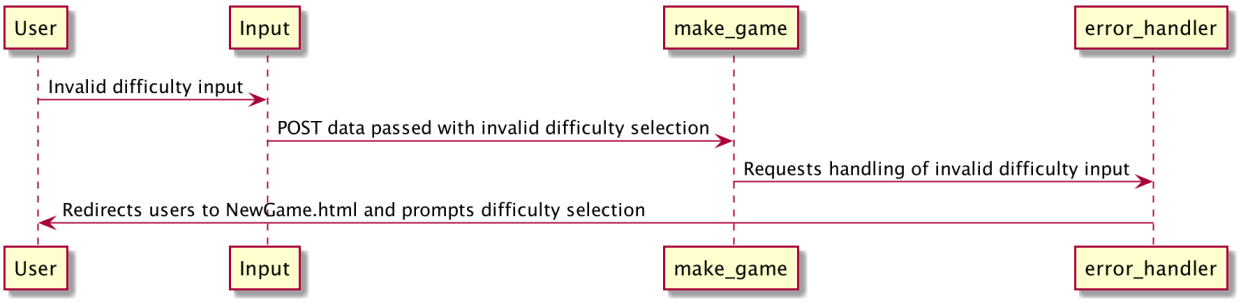

Figure 4 - Scenario 3: Error Handling – Invalid Difficulty

Scenario 4: Error-handling—Invalid user name entry

Description: Username is both stored in the database and redisplayed in HTML, requiring sanitition to avoid multiple forms of injection attacks. An unsafe user name entry forces the application to store the name as "Anonymous."

Pre-condition: User visited the home page, clicked Start a New Game, and newgame.html has been rendered.

Post-condition: A board is created, but play.html displays "Anonymous" as the user name.

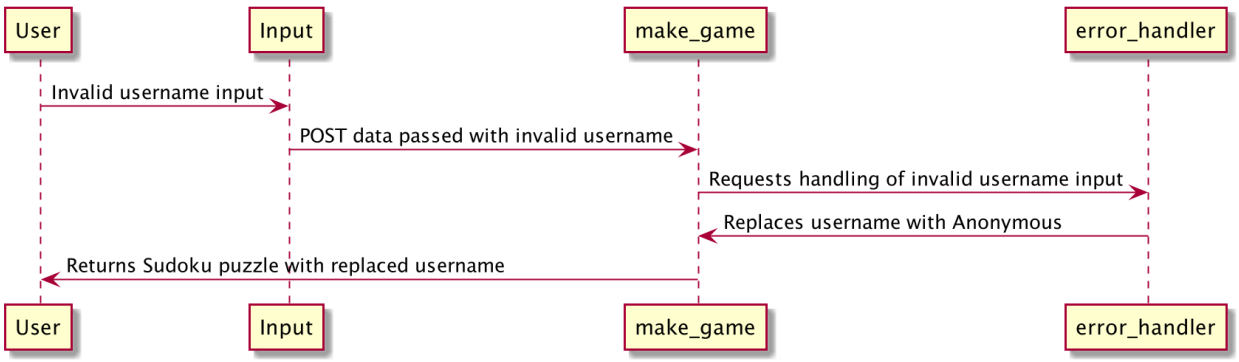

Figure 5 - Scenario 4: Error Handling – Invalid Username

~~**Scenario 5: Error-handling—Database connection error**~~

~~Description: Active game play can be successfully completed with only the JavaScript session data. Django session data affords the user the opportunity to refresh the browser without losing the state of the game. The database is used for long term retention of game play statistics. In the event of database connection issues, the web app should gracefully continue, allowing the player to continue, with a notification to support staff to troubleshoot the issue.~~

~~Pre-condition: User is in-process of playing a game.~~

~~Post-condition: Support staff notified of issue, user completed the puzzle, notified of failure to save leaderboard information.~~

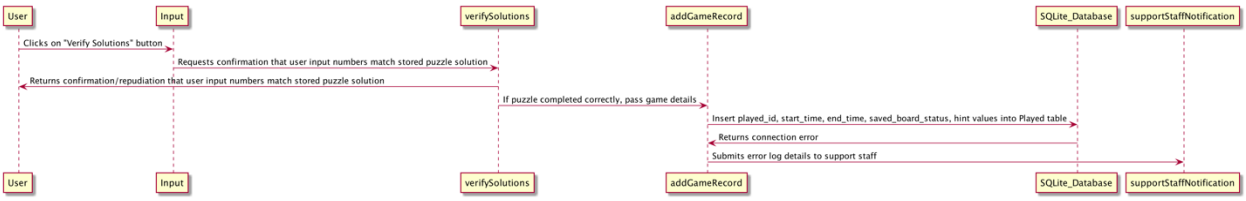

Figure 6 - Error Handling - DB Connection

Scenario 6: Error-handling—Intentional or unintentional termination of application

Description: The current state of the game puzzle is saved in both the JavaScript session data, the Django session data, and the SQLite3 database, every time the scratch pad is closed or associated with a different puzzle cell. In the event a user accidentally exits the browser window and clears their session data, they should have the opportunity to resume the game by loading it from the database.

Pre-condition: The user has started a game and made progress towards a solution.

Post-condition: The user returned to the website and successfully loaded the previous state of the board to continue playing.

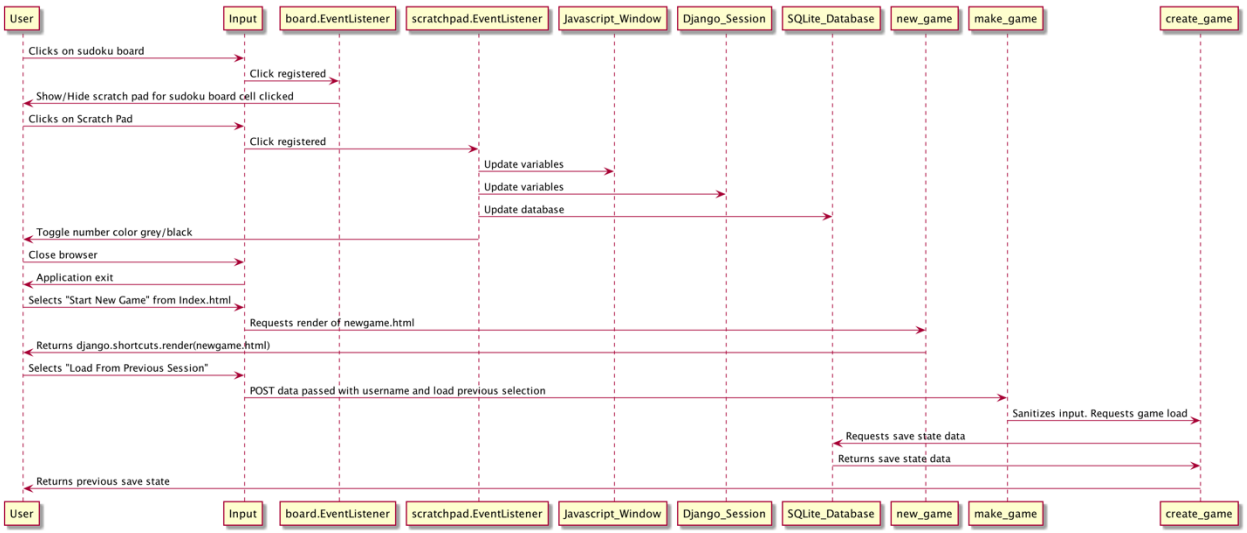

Figure 7 - Scenario 6: Unintentional Termination

Scenario 7: Visitor views all static informational webpages

Description: A user enters our URL into their browser and arrives at our home page, reads the static content, and then visits each of the pages to see what is presented.

Requirements Addressed: SUD-5, 6, 7, 8, 19, 28, 29, 30, 31

Pre-condition: Webserver is running, ready to accept incoming connections.

Post-condition: Static information from each of the requirements addressed, and depicted in diagram below, have been displayed in the users web browser.

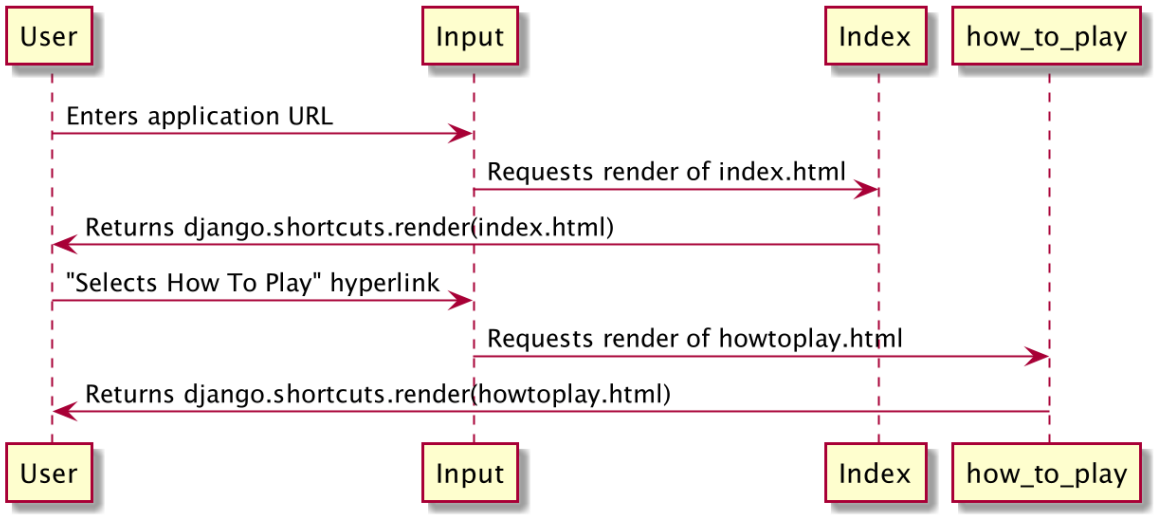

Figure 8 - Scenario 7: Visitor views static webpages

Scenario 8: Visitor initiates a new game

Description: Player requests to start a new game, enters their name and desired difficulty, and plays the game to completion.

Requirements Addressed: SUD-9, 10, 11, 13, 14, 15, 16, 17, 18, 24, 25, 26, 34, 45, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58.

Pre-condition: Player visited, and was served, our homepage for display in their web browser.

Post-condition: Player has completed game play and either solved the puzzle correctly or incorrectly, with a message output to the users screen confirming if their solution is correct or not.

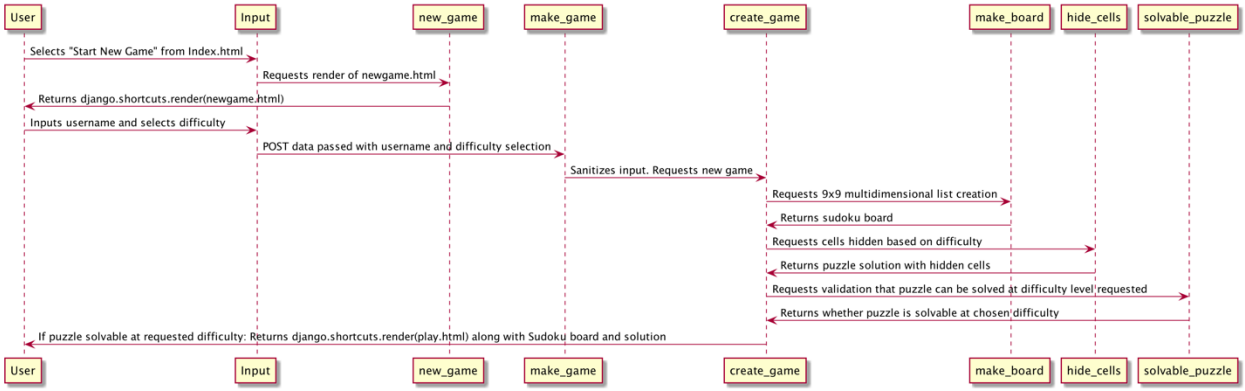

Figure 9 - Scenario 8: Visitor initiates new game

Scenario 9: Scratchpad number removal

Description: Player clicks on an unsolved cell in the puzzle, which opens the scratchpad. One or more numbers are clicked in the scratchpad to remove them.

Requirements Addressed: SUD-9, 22, 34, 35, 36, 37, 38, 39, 40, 41, 42, 58, 60, 61.

Pre-condition: Player initiates a new game.

Post-condition: Player has successfully removed number(s) from an unsolved cell using the scratchpad.

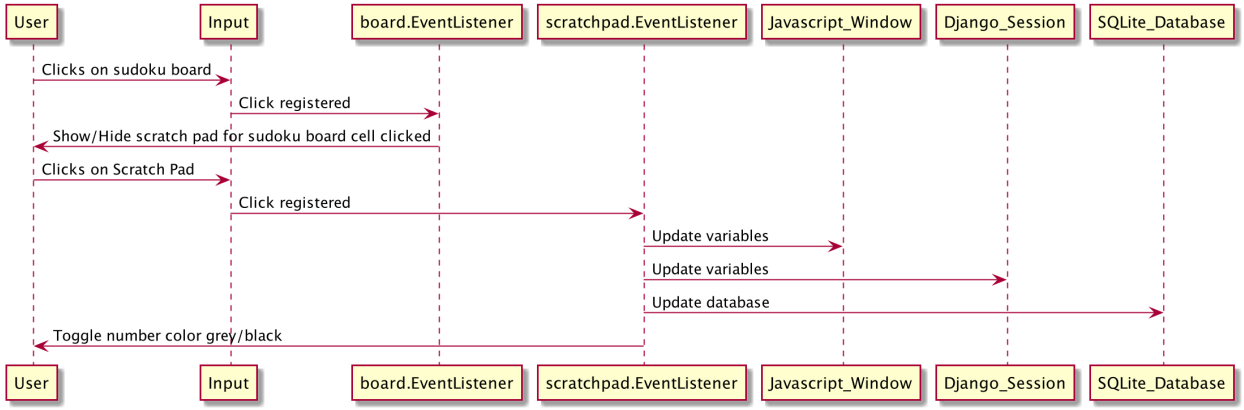

Figure 10 - Scenario 9: Visitor plays a new game

Scenario 10: Solve a single cell

Description: Player removes all but one number from the scratchpad, which displays a blue number of equal size to the given black numbers. Player then clicks on 'Verify Solution' and is notified if the puzzle contains an error or if they are still on track to solve the puzzle.

Requirements Addressed: SUD-9, 22, 34, 35, 36, 37, 38, 39, 40, 41, 42, 58, 60, 61

Pre-condition: Player has initiated a new game.

Post-condition: A cell previously containing 9 candidates now displays a single, full-size, blue number to represent the user's solution for that cell. Clicking 'Verify Solution' presented a message indicating a correct or incorrect solution.

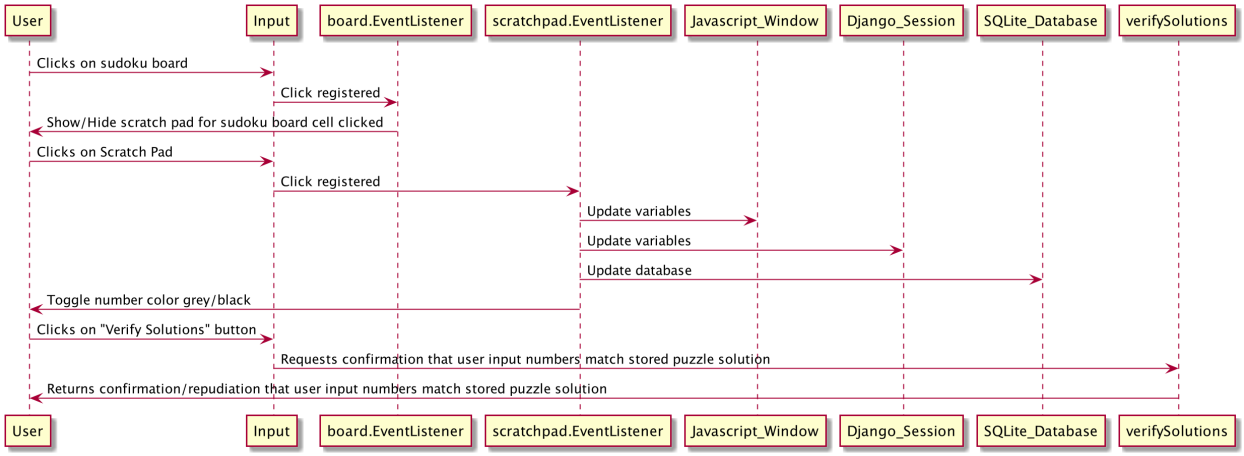

Figure 11 - Scenario 10: Solved Single Cell

Scenario 11: Solve entire puzzle

Description: Player solves the last remaining cell and clicks 'Verify Solutions'.

Requirements Addressed: SUD-9, 22, 34, 35, 36, 37, 38, 39, 40, 41, 42, 58, 60, 61

Pre-condition: Player initiated a new game and solved all unknown cells except for one.

Post-condition: Player successfully solves the last cell, is alerted if successful or failed, and results are stored in the database for use in leaderboard display.

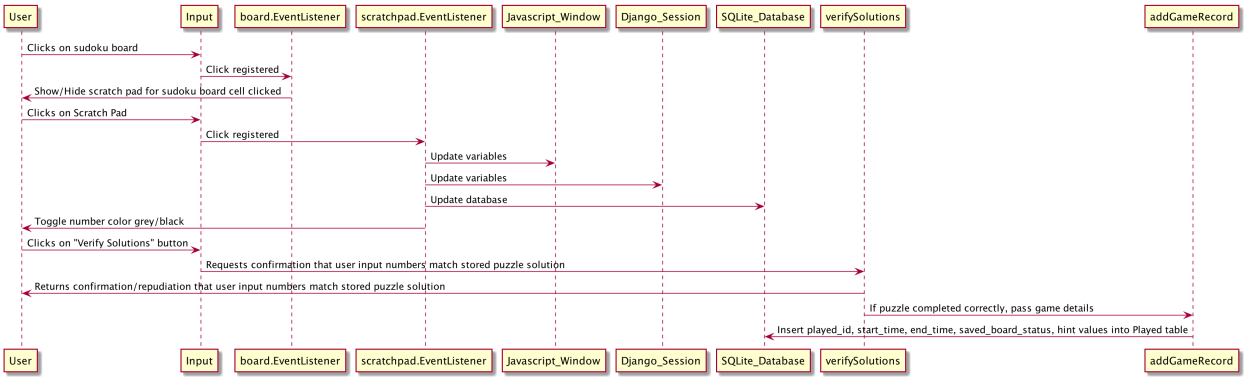

Figure 12 - Scenario 11: Solve Puzzle

Scenario 12: Visitor visits the leaderboard and challenges a top score

Description: Player requests to view the leaderboard, which displays the fastest 5 successful puzzle solutions for each of the 5 difficulty levels. After viewing these statistics, the player choses one of the puzzles from the leaderboard to play, in an attempt to achieve a better time.

Requirements Addressed: SUD-32, 33, 44.

Pre-condition: At least 5 games at each difficulty level have been played by the developers and stored in the database for display on the leaderboard. Visitor is currently viewing the homepage.

Post-condition: Player successful started a new game using a puzzle from the leaderboard.

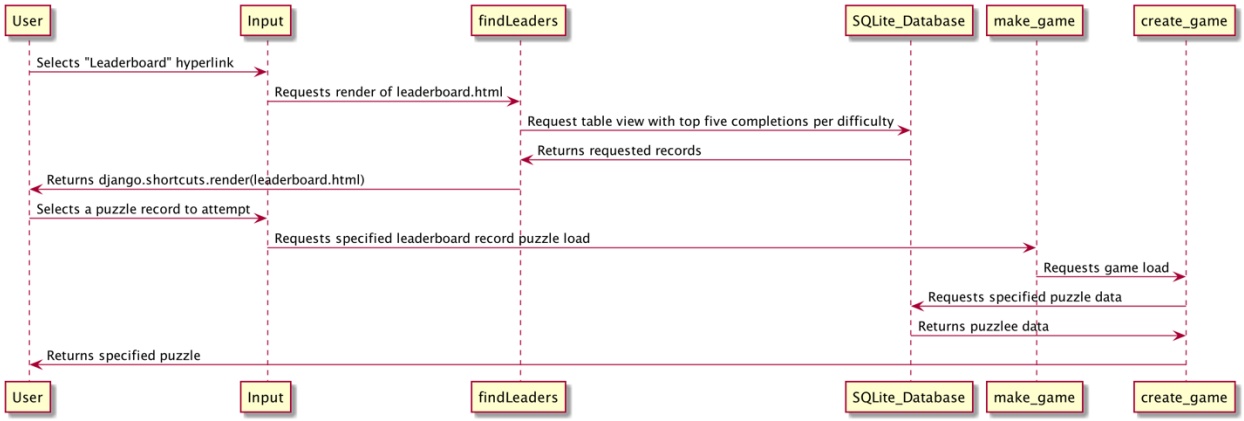

Figure 13 - Scenario 12: Visitor challenges top score from leaderboard

Scenario 13: Visitor requests to play a custom puzzle of their own

Description: Player has their own puzzle from an outside source such as newspaper, game book, another website, etc. The player wishes to manually enter the starting board of the puzzle, and play the game on our website.

Requirements Addressed: SUD-46, 47, 48.

Pre-condition: Visitor is currently viewing our home page.

Post-condition: Player has entered the starting value (or lack thereof) for each cell on the puzzle, it has been validated to have only one unique solution, and the player is presented with the game board, ready to be played.

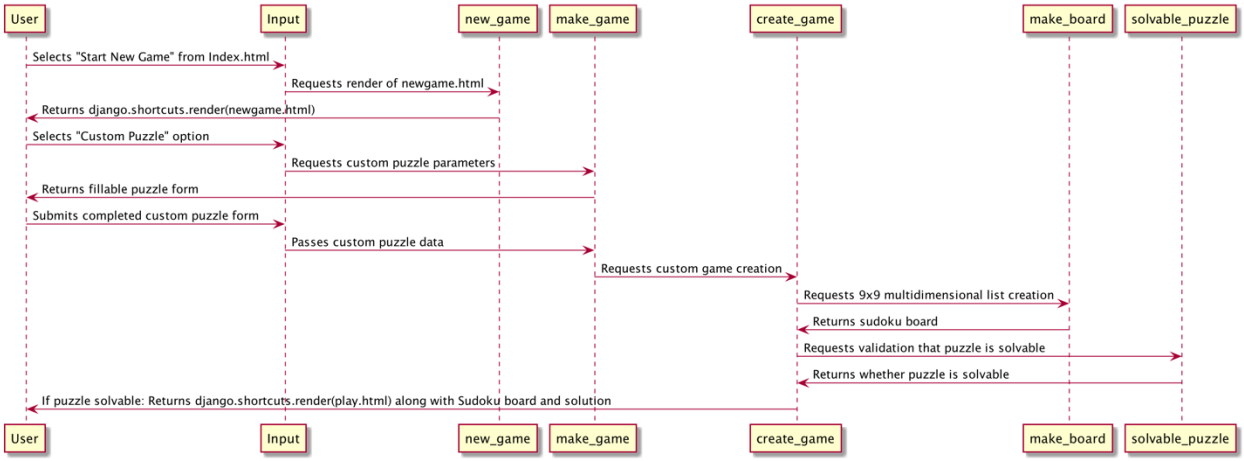

Figure 14 - Scenario 13: Custom Puzzle

Scenario 14: Administrator uploads locally generated puzzles into the database

Description. User with administrator credentials can visit a webpage to upload locally generated puzzles into the database.

Requirements Addressed: SUD-63

Pre-condition: Web server running, administrator knows credentials and non-linked upload URL.

Post-condition: Database contains new puzzles from the uploaded file.

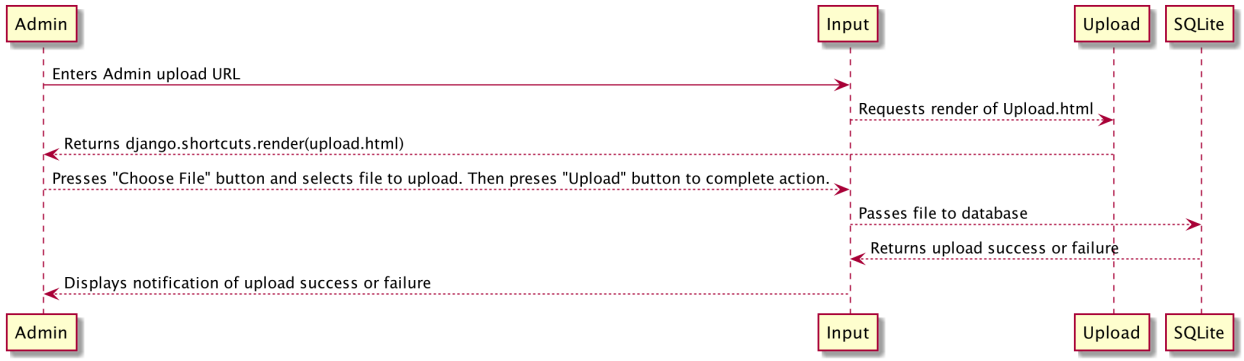

Figure 15 - Scenario 14: Admin Upload

Scenario 15: Player requests a hint

Description. Player is stuck, unable to figure out the next move, and wishes to get a hint.

Requirements Addressed: SUD-65

Pre-condition: Player has started a new game and actively playing.

Post-condition: The name of a technique, and general location (row, column, block) where to apply it, is presented to the user on-screen.

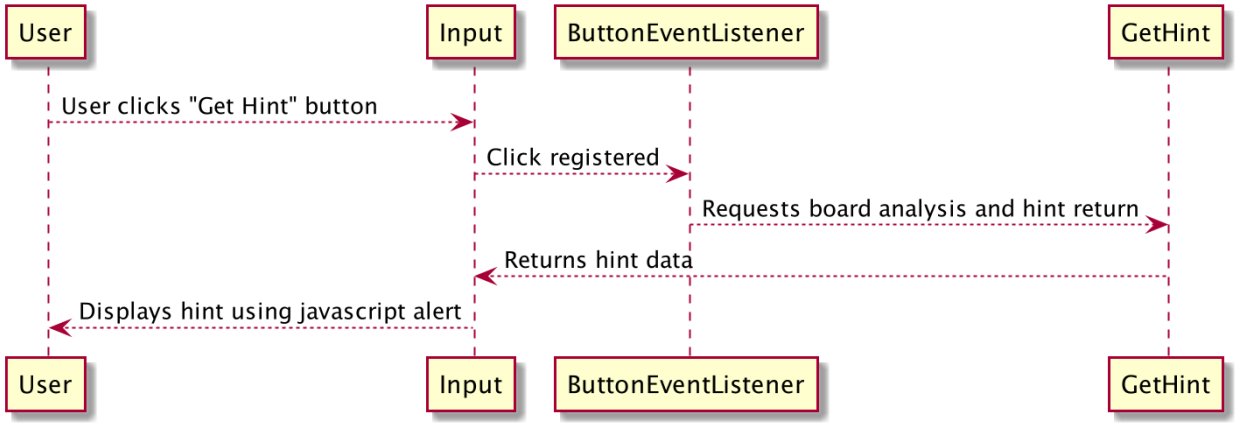

Figure 16 - Scenario 15: Get Hint

~~CLASS~~ FUNCTION DESIGN (Our Python Web App will not be using classes)

Input Subsystem

```
views.py {
    def index(request):
        return django.shortcuts.render(index.html)
    def how_to_play(request):
        return django.shortcuts.render(howtoplay.html)
    def leaderboard(request):
        return django.shortcuts.render(leaderboard.html)
    def new_game(request):
        return django.shortcuts.render(newgame.html)
    def play(request):
        return django.shortcuts.render(play.html)
}
```

NewGame Subsystem

```
newgame.html {
    # html web form to accept and pass player name and difficulty level to the backend
    POST name and difficulty to make_game()
}
```

CreateGame Subsystem

```
create_game.py {

    def create_game(difficulty):
        while True:
            while True:
                solution = make_board()
```

```
            # make_board() attempts to randomly create a valid puzzle
              solution and return it, or false if it fails.  Loop until it succeeds.
        board = hide_cells(solution, difficulty)
        # hide_cells randomly removes solved digits, quantity based on difficulty level.
        if solvable_puzzle(board, difficulty):
            return board, solution
        # solvable_puzzle attempts to solve the board using techniques of selected difficulty


def make_board():
    board = 9x9 multi-dimensional list
    for row in range(9):
        for col in range(9):
            calculate list of numbers allowed in the current row, col based on sudoku rules.
            board[row][col] = random number from allowed list
    return board


def hide_cells(solution, difficulty):
    hide_count = random number of cells to hide in the solution, based on difficulty level.
    while counter < hide_count:
        randomly choose a cell.
        replace the solution number with nested list of numbers 1-9 to represent pencil marks.
    return original solution with nested [1-9] in cells player must solve themselves.


def solvable_puzzle(board, difficulty):
    while (solving techniques continue to remove pencil marks):
        naked_single
        hidden_single
        if difficulty > 1:
            naked_pair()
            omission()
            naked_triplet()
```

```
            if difficulty > 2:
                hidden_pair()
                naked_quad()
                hidden_triplet()
            if difficulty > 3:
                hidden_quad()
                xwing()
                swordfish()
                xywing()
                unique_rectangle()

        # techniques no longer able to make progress
        if is_solved(puzzle):
            return True
        else:
            return False

    def is_solved(puzzle):
        loop through all puzzle cells
            if cell is a list  # indicates pencil marks remain and it wasn't completely solved
                return False
        return True
}

views.py {
    def make_game(request):
        Get POST data for player name and difficulty level, then sanitize it.
        If data is valid:
            new_board, solution = create_game(difficulty)
            Add new_board and solution to Django session data
            HttpResponseRedirect play(request)
```

```
        else:
            error handler
}
```

Play Subsystem

```
play.html {
    # JavaScript front-end code to enable game play interaction with user
    welcome the player
    drawBoard() # draw puzzle canvas
    drawNumbers() # populate the puzzle with the partial solution
    showPencilmarks() # display [1-9] in cells whose solution must be deteremined by user

    board.EventListener('click') {
        if mouse clicked:
            get row and cell where user clicked
            if scratch pad is already visible and associated with same cell:
                hide the scratchpad
                redraw available numbers in cell on puzzle board
                remove highlighting around active cell
            elif scratch pad not visible or visible but associated with different cell:
                if scratchpad was visible on previous cell:
                    remove highlighting around active cell
                    redraw available numbers in cell on puzzle board
                show the scratchpad
                update scratchpad numbers colors; black in play, grey removed
                highlight active cell scratchpad is associated with
    }

    scratchpad.EventListener('click') {
        if mouse clicked:
```

```
            get row where user clicked
            toggle color (grey/black)
            update JavaScript window variables
            update Django Session variables
            update SQLite Database
    }

    function verifySolutions() {
        for (var row = 0; row < 9; row++) {
            for (var col = 0; col < 9; col++) {
                if board[row][col] is a list of length 1 {
                    compare solution[row][col] with board[row][col]
                    if not equal, return false
                } else {
                    cell was not fully solved (more than one pencil mark remains); return false
                }
            }
        }
    }
}
```

GameRecords Subsystem

```
gamerecords.py {
    def addPlayerRecord(various Played columns listed below) {
        connect to the database
        define played_id, puzzle_id, start_time, end_time, saved_board, status, hints
        insert these values into the Played table
}

def addGameRecord(various Puzzles columns listed below):
    connect to the database
```

```
    define puzzle_id, board, solution, difficulty, naked_single, hidden_single, naked_pair,
    omission, naked_triplet, hidden_pair, naked_quad, x_wing, swordfish, xy_wing,
    unique_rectangle
    insert these values into the Puzzles table

leaderboard.py {
    # this function be called in newgame.html for playing saved games
    def retrieveGame(puzzle_id):
        connect to the database
        # uses user-entered puzzle_id
        select corresponding board from Puzzles
        return board
}
```

Leaderboard Subsystem

```
leaderboard.py {
    def findLeaders():
        connect to the database
        create a view for the difference in start and end times paired with difficulty
        sort this view in ascending order and select the first five users for each difficulty
        return these users with calculated completion times to be displayed on leaderboard.html
}
```

UserPageSelection Subsystem

```
    index.html {
        # static html page displays information about the project and developer team
    }
    howtoplay.html {
        # static html page displays a brief history, rules of the game, and an explanation of the two
            beginner techniques: hidden single and naked single.
    }
```

UNRESOLVED RISKS AND RISK MITIGATIONS

- The web application may be vulnerable to CSRF attacks between various pages. To remedy this, an anti-CSRF token should be implemented per session to prevent malicious transmission of code.
- Vulnerability to SQL injection. Mitigating this risk involved using parameterized queries of whitelisted characters to ensure that the database is not compromised.
- Vulnerability to general code injection in username textbox. Akin to risk #2, whitelisting characters are vital in preventing a would-be malicious user from injecting code through the textbox. While credential safeguarding is not an issue in this application, performance issues may arise without preventing unexpected character entry.
- Chrome users may encounter an issue regarding cross-site cookies that lack a SameSite attribute related to the application's LinkedIn badges. This warning stems from recent configuration changes on Chrome's end pertaining to how cookies are handled between different websites.