ElijahC Canoga Java/Android Manual

Bug Report

To my knowledge, there are no major bugs in the program. It performed all game functions as I expected throughout my testing. However, below is a list of minor bugs that are worth noting.

- *Player makeMove()*: When a player wins on the first turn by covering all of their own squares, the program does not suggest a winning move the same way it typically does (by announcing that a winning move was found). It instead suggests the winning move as an optimal move

- *Player makeMove()*: When a player wins on the first turn by covering all of their own squares, they are prompted to roll the dice again and take an extra move, even though they have already technically won. Since there are no moves available, the turn ends anyway and the win is declared correctly. However, an extra move is prompted by the program.

- GameActivity setUpBoard(): The setUpBoard() function is not called when the user changes the orientation of their device. Therefore, the orientation keeps the previous board layout. This means that if you switch from horizontal to vertical mid-game, the squares will remain in a single row, and if you switch from vertical to horizontal mid-game, the squares will remain in rows of 3. This bug only occurs when switching orientation mid game. This bug does not affect gameplay, only display.

Feature Report

Missing Features:

- *No major features are missing (every feature explicitly listed in the project guidelines is implemented)*
- Saving the game does not automatically exit the game. This was intentional, as you can then save at various points within the same game. You can always click quit right after saving to leave.
- Not sure if this is a missing feature, but there are no Tournament or Round classes in my code. Round is basically CanogaModel, and Tournament is circumvented by the button logic in CanogaController
- I tried to follow the MVC design pattern the best I could, but there are some areas that are potentially debatable, namely the log mechanism and the update of the board size. However, nothing in the model package receives any Android imports whatsoever (and is never passed any Android component or reference to the view), so MVC design is ultimately preserved. Just minor deviations, if any.

Extra Features:

- Board switches layouts based on device orientation (grouped in rows of 3 for portrait/vertical, one single scrollable row for horizontal/landscape)
- Game log is available in both the main game and the post game results
- When the human moves, squares highlight blue for covering and green for uncovering

Data Structures and Classes

Core Data Structures

- ArrayList<Integer>
    - Used throughout to hold the current values of the squares (1…N or 0 for covered), lists of available square values, and collections of moves (each move is itself an ArrayList<Integer>).
- ArrayList<ArrayList<Integer>>
    - Represents the full set of valid move subsets (size 1–4) that sum to a given dice roll.
- HashSet<Integer>
    - Tracks user's current square selections (either human or computer) during move selection to prevent duplicates.
- StringBuilder
    - Efficiently accumulates text for log messages and board-printing routines.

Class Hierarchy & Composition

Board

- **Responsibility**: Maintains the numeric state of each player's row of squares and the current turn count.

- **Members**:
    - int size
    - ArrayList<Integer> humanSquares
    - ArrayList<Integer> computerSquares
    - int turn
- **Composition**: Used by both Player subclasses and by the model/controller to enforce game rules and queries (e.g. “all covered?”, “available squares for cover/uncover?”).

BoardView

- **Responsibility**: Formats and logs a text-based display of the two rows (computer & human).
- **Members**:
    - Board board
    - Log log (interface)

Log (interface)

- Single method logMessage(String) used by all model/controller classes to append to the game log without knowing UI details.

Player ← *abstract base class*

- **Responsibility**: Encapsulates shared state (round score, tournament score, turn flags, handicap) and common logic (move-generation,

instant-win search, strategy helpers).

- **Members** (protected/private): roundScore, tournamentScore, isFirst/isNext flags, wonByCover, handicapSquare, etc.

- **Key methods**:

    - chooseNumDice(...), allValidMoves(...), getInstantWinMove(...), shouldCoverOwnSquares(...), and abstract rollDice(...) & makeMove(...).

- **Inheritance**: extended by → **Human** and **Computer**

Human & Computer

- **Responsibility**: Provide concrete implementations of rollDice(...) and makeMove(...) (including UI- vs. AI-driven logic).

- **Distinctions**:

    - Human logs user prompts and enforces valid input.

    - Computer uses shouldCoverOwnSquares etc. to choose moves and prints rationale.

CanogaModel

- **Responsibility**: Handles core game flow (load/save serialization, first-player determination, handicap, round scoring, alternating turns).

- **Composition**:

    - Holds a Board, one Human, one Computer, and a Log.

- **Key data structures**:
    - ArrayList<Integer> for reading/writing board state in files.
    - StringTokenizer / I/O streams for serialization.

CanogaController

- **Responsibility**: Implements the finite-state machine driving UI interaction (“START_GAME”, “NUM_DICE”, “MOVE_SELECTION”, etc.).
- **Composition**:
    - Holds a CanogaModel and a CanogaView reference.
- **Key structures**:
    - ArrayList<Integer> userManualRolls
    - HashSet<Integer> selectedHumanSquares, selectedComputerSquares
    - Tracks currentState, userNumDice, userDiceSum, userCoverChoice.

GameActivity (CanogaView)

- **Responsibility**: Android UI layer—renders the board in two GridLayouts, three “dynamic” buttons, score/log TextViews, and dialogs for manual rolls.
- **Inheritance**: extends AppCompatActivity, implements CanogaView.

- **Composition**:
    - Holds a CanogaController and all UI widgets.
    - Maintains an ArrayList<Integer> originalSquares to remember the labels 1…N for selection highlights.

Other Activities

- **MainMenuActivity**, **BoardSizeActivity**, **FileLoadingActivity**, **TournamentResultsActivity**
    - Each extends AppCompatActivity, handles navigation and minimal logic (e.g. listing saved .txt files in FileLoadingActivity using getFilesDir() + asset copies).

Inheritance & Composition Summary

- Inheritance
    - Player → Human & Computer
    - Android screens all extend AppCompatActivity
    - GameActivity implements CanogaView

- Composition
    - CanogaModel contains Board, Human, Computer, Log
    - CanogaController contains CanogaModel, CanogaView, plus selection & roll-tracking collections
    - GameActivity contains CanogaController and all the UI widgets

Progress Log

March 21, 2025

- **Started up Project (1 hour)**
    - Created project directory and created .java files for 7 main classes (Board, BoardView, Human, Player, Computer, Round, Tournament), as well as main file called Canoga
    - Tested the transfer of data between files and basic fundamentals of java (being able to run the program in terminal successfully)

March 22, 2025

- **Implemented Java version of C++ Board class (2 hours)**
    - Used ArrayList in place of vector for storing board data
    - Created set and get functions for all Board data members, just as in C++
    - Mimicked Board functions such as resetBoard(), checkUpperSquares(), and uncoverSquare() in java

March 23, 2025

- **Consolidated code and Implemented BoardView class (1 hour)**
    - Translated BoardView class and printBoard() functions from C++ version to java
    - In Canoga.java, created board and view objects and successfully printed the board to the terminal, just as in C++

March 26, 2025

- **Helper Classes and Player Class (2 hours)**
    - Created an “Input” helper class to handle input similar to cin in C++. This helper makes it easier in other files to read input, as the object has already been created
    - Created a “Pair” helper class to act similarly to a pair data type in C++. This pair class is used when reading dice roll sequences, and allows me to preserve my code stye from my previous game in Java
    - Added member variables to track player data throughout the game
    - Created set and get functions for each member variable in Player

March 27, 2025

- **Completed Player Class translation (2 hours)**
    - Defined abstract functions rollDice() and makeMove() to be used by Human and Computer classes
    - Implemented all player functions from C++ version, using Java syntax. ArrayList is used in place of vectors. Custom pair class used for pair data type.
    - Created helper function copyBoard() in order to simulate winning move on a future board state

March 28, 2025

- **Completed Human Class translation (2 hours)**
    - Defined constructor using super, as Human inherits from Player
    - Implemented both overridden classes rollDice() and makeMove()
    - Used Input helper class to handle all human input reads

- **Completed Computer Class Translation (2 hours)**
    - Defined constructor using super, as Computer inherits from Player
    - Implemented both overridden classes rollDice() and makeMove()
    - Used Input helper class to handle all human input reads

March 29, 2025

- **Completed Round Class Translation (2 hours)**
    - Added all necessary member variables for round class, and created constructor that reads in board, human, and computer objects
    - Translated all Round functions that control game logic and changed necessary syntax
    - For serialization functions, used Java's file io objects and methods to control serialization flow
    - Used classes like BufferedReader, FileReader, PrintWriter, and FileWriter to handle file reading and writing.
    - Parsed text files using StringTokenizer, to replace custom token parsing logic from C++ version

- Used an ArrayList to store player squares and parsed scores with Integer.parseInt()

March 30, 2025

- **Completed Tournament Class Translation (2 hours)**
  - Translated Tournament member variables and functions just as in C++, changed necessary syntax
  - startTournament() starts a game from scratch, resumeTournament() starts with a serialization game, and displayTournamentResults() displays the end result of the Tournament, just as in C++ version

March 31, 2025

- **Created Main Canoga Implementation, Fixed Bugs (2 hours)**
  - Created implementation for main game file, with 3 options for user (start new game, load game from serialization file, exit program), just as in C++ version
  - Successfully ran the pure Java version of Canoga in the terminal. Was able to start new game, load from serialization, and play the game normally
  - Fixed minor bugs in each of the game files. TurnActive in round needed an initialization. validMoves() function in Player class needed additional logic within the nested for loops for adding moves to the list of valid moves.
  - The program now behaves just like my final C++ Canoga implementation

April 3, 2025

- **Set up Android Studio Project (2 hours)**
  - Created a new project in Android studio to host the Android GUI version of Canoga, using the Empty Activity Template

- Created a model package to store my previous model code, and left the main package for GUI operations
- Planned out the activity flow for the game: there would be a welcome activity, a board size selection activity, a file selection activity, a main game activity, and a tournament results activity.
- Created files for each of these activities, and gave basic navigations and buttons to go from activity to activity
- App compiled successfully, and routes were handled successfully from activity to activity

April 5, 2025

- **Designing Visual interfaces for each activity (2 hours)**
  - MainMenuActivity: Added a header and 3 buttons to the welcome screen. The first button leads to board selection, the second leads to file selection, and the third exits the app.
  - BoardSizeActivity: Added a header, 3 board size option buttons (9, 10, 11), and a back button. Each of the board size buttons bring the user to the main game activity, with an added “putExtra” parameter listing the desired boardsize
  - TournamentResultsActivity: Added a header, a space for the tournament results to be shown, a Main Menu button, and an exit button.
  - GameActivity: Added a header and a place for scores to be shown. Under that, a section for the game log to be shown. Under that, a space for the board for both players to be shown, and under that, a place for the main game buttons to be shown.
  - All corresponding XML files were updated to reflect these changes

April 6, 2025

- **Introduced Game Flow to GameActivity (2 hours)**
  - Planned out method of handling event-driven design: the game would rely on states. Each state would correspond to an action the program must take in response to user input. In this case, we are only worrying about managing buttons, but later it will be used to manage the entire game
  - Created an Enum GameState that holds the state of the game: StartGame, BeginTurn, NumDice, RollType, MoveType, MoveSelection, and RoundEnd.

- The buttons now change their visible text and availability based on the current gamestate, and change the state when pressed to the next logical state. For simplicity I gave them the order they were created in, just to test functionality. Later, this will be determined by actions taken by the actual model
- Dynamic buttons 1, 2, 3 are controlled by handleDynamicButtonClick(), and the UI update is controlled by updateDynamicButtons. The save and quit buttons are persistent and do not change based on game state. You can always save and always end the tournament

April 7, 2025

- **Modifying Model Code to be Event-Driven (2 hours)**
  - Created a ConogaModel class to be the main model for the game
  - Deleted the Round and Tournament classes (no longer necessary for event-driven design)
  - Added save and loadGame functions from previous Round class to CanogaModel
  - Added determineFirstPlayer(), alternateFirstPlayer(), determineHandicap(), and sumOfTwoDigits() from previous Round class to CanogaModel
  - Created functions to handle user actions, such as startGame(), numDiceHelp(), moveTypeHelp(), moveSelectionHelp(), and roundEnd(), taken from previous console-based game
  - Created humanMove() function to handle when the human player submits squares, which gets passed as a parameter a set of the selected squares
  - Created a computerMove() function to exhaust all possible moves that turn, similar to the console implementation
  - For now, all moves rely on random rolls, with the inherited rollDice() function, and still used the inherited makeMove() function to actually implement the move to the internal board

April 9, 2025

- **Creating interface between model and view (1 hour)**
  - Created CanogaUI interface to manage the output of game messages
  - Implemented the UI in the view so that log messages are displayed to the scrollable game log
  - Changed all console outputs to utilize the ui.logMessage() function in the interface instead of printing the line to the terminal

April 11, 2025

- **Implementing Baseline GUI Canoga game (2 hours)**
    - Modified the updateBoardDisplay() function in GameActivity to dynamically update based on the board state in the model when called. Also allows users to click on squares, and those squares light up when selected, and record them in an array selectedSquares for later processing
    - Used model functions to direct gameState in handleDynamicButtonClick
        - **StartGame**: choose the first player and switch to BeginTurn
        - **BeginTurn**: If it's the human's turn, we check if they can choose 1 or 2 dice and direct them accordingly. If it's the computer's turn, we go straight to rollType
        - **NumDice**: Allows the user to select 1 or 2 dice to roll, with an option for help
        - **RollType**: Allows the user to select random or manual rolls. For now, this is an intermediary state, as there is no manual dice yet
        - **MoveType**: User selected cover or uncover move type, with an option for help
        - **MoveSelection**: user selects the squares they want to cover/uncover, and those squares get added to selectedSquares array. The buttons allow users to clear their selection, submit the move, and ask for help
        - **RoundEnd**: If a winner is detected, we end the round. If the user starts a new round, we start over at StartGame. If not, we go to the Tournament screen
    - TournamentResultsActivity now receives parameters for the human score, computer score, and the gameLog for display and evaluation.
    - Game is now playable with random dice rolls only

April 13, 2025

- **Implementing File Loading and Serialization Plays (2 hours)**
    - Saved serialization files to internal memory of the Android device
    - Created a function to list all .txt files in the internal memory location of the app with getFilesDir()
    - Each list element is a button that stores that filename as the filename to be passed to the GameActivity

- The filename is passed as a parameter to the GameActivity, to later be used by model.loadGame()
- If GameActivity receives a filename in its parameters, we initiate model.loadGame(). If not, we created a new game from scratch
- Serialization files are now playable with random rolls only

April 15, 2025

- **Separating Code into MVC Design (2 hours)**
  - Created GameController file to manage game state, in order to decouple controller responsibilities from the view (GameActivity)
  - Moved all Enum operations to this file, and updated function names accordingly
  - Moved saved game loading to the Controller
  - Changed updateBoardDisplay() to rely on Controller functions to get necessary values (such as square values for the players)
  - Removed all references to the model in the view, including human, computer, and board object creations. View now only creates a controller, and has no model object or sub-objects within the model.

April 16, 2025

- **Removing GameActivity from Model (2 hours)**
  - Deleted previous CanogaUI interface, as that was implemented by GameActivity and was used directly by the model and model functions. Not MVC Design.
  - Created two new interfaces: CanogaView and Log
  - CanogaView is the actual view interface. This links the controller to the view so that we can refresh the view when needed
  - Log is the interface that logs game messages. This is implemented by the Controller only.
  - Now, the Model has no reference to the view, and the model is not accessed by the view. The Controller acts as the intermediary between them, and the two are not passed to each other as parameters whatsoever.
  - logMessage() now passes the message to Game Controller, which then passes the message to GameActivity's displayMessage() function. Model has no knowledge of the view. View has no knowledge of the model.

April 17, 2025

- **UI Updates and Usability Improvements (2 hours)**
    - Added log button for full game Log display
    - showGameDialog() creates a dismissable alert that allows the user to scroll through all game messages and see what has transpired in the game
    - Fixed button formatting in activity_game.xml so that buttons and UI have better spacing and color scheme
    - Created different square layouts for horizontal and vertical screen orientations (1 row for horizontal, rows of 3 for vertical)
    - Fixed up all xml files to have better spacing and sizing

April 18, 2025

- **Manual Roll Implementation (2 hours)**
    - Created two alerts in GameActivity to handle collecting user input for manual dice rolls
    - The first alert in promptForManualRollCount() asks the user how many dice rolls they would like to enter, with list options 1-10 (up to 10 manual dice rolls at a time, and the user can always enter more after those are used up)
    - The second alert in promptForManualDiceValues() asks the user to input 2 dice values per roll, representing the two dice. These are stored in an array list that is then passed to the controller for use
    - The controller then passes the rolls to the computer if it is the computer's turn, or it calls the function popManualRollOrRandom() when it's the human's turn. This way, the computer can do its moves automatically, while the human player can go one by one
    - Game can now be played using manual rolls, and serialization files can be tested

April 19, 2025

- **Adding Board size selection and additional Clean up (2 hours)**

    - Board size is now prompted for at the end of each round
    - In game board displays are properly updated, and model adjusts accordingly
    - Fixed bug where board reset to a lower size would crash the game, which is prevented by clearing the current views before implementing the new ones
    - Fixed bug where game would reset when screen orientation was changed. Updated manifest to prevent destruction and rerendering of screen on orientation change
    - Allowed horizontal scrolling of board squares so square 11 would be more visible
    - Implemented nested scroll views so that game log scroll view would not interfere with main scrollview

Total Time Spent on project:
~ 45 hours (~50 hours with documentation and write-ups)

Help from External Sources

- The code in my program is my own code, and there are no significant references

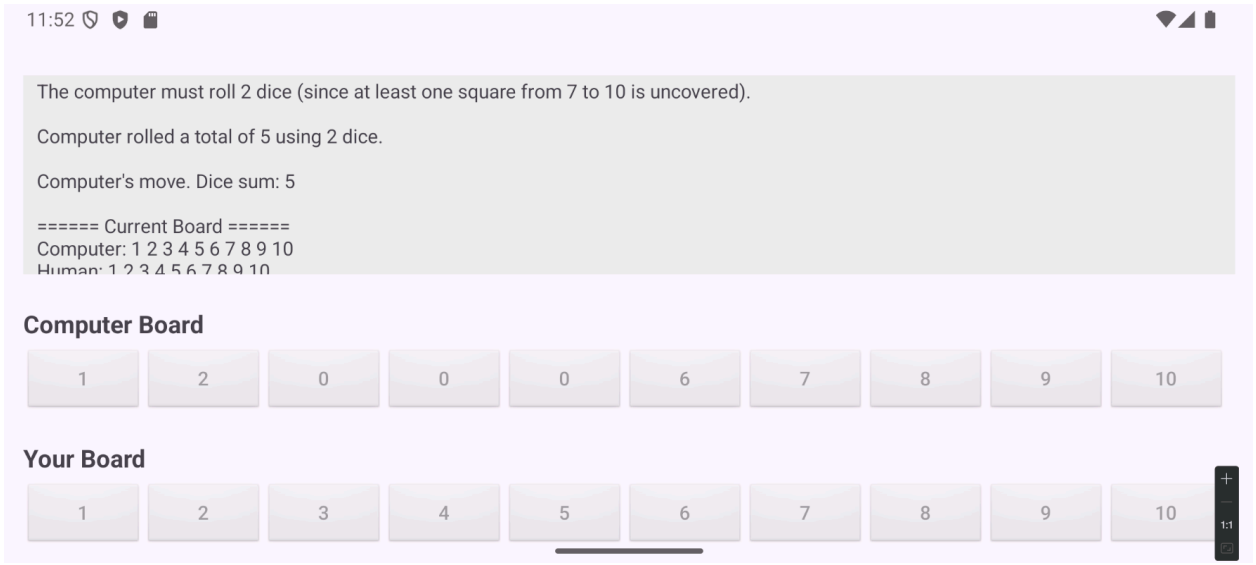
- I was unaware of how the file system worked within the internal storage of the android device emulator. The function copyFilesFromDefaultAssets() is inspired by AI generation (namely Gemini), and I edited it to my needs (to include the 5 serialization files)

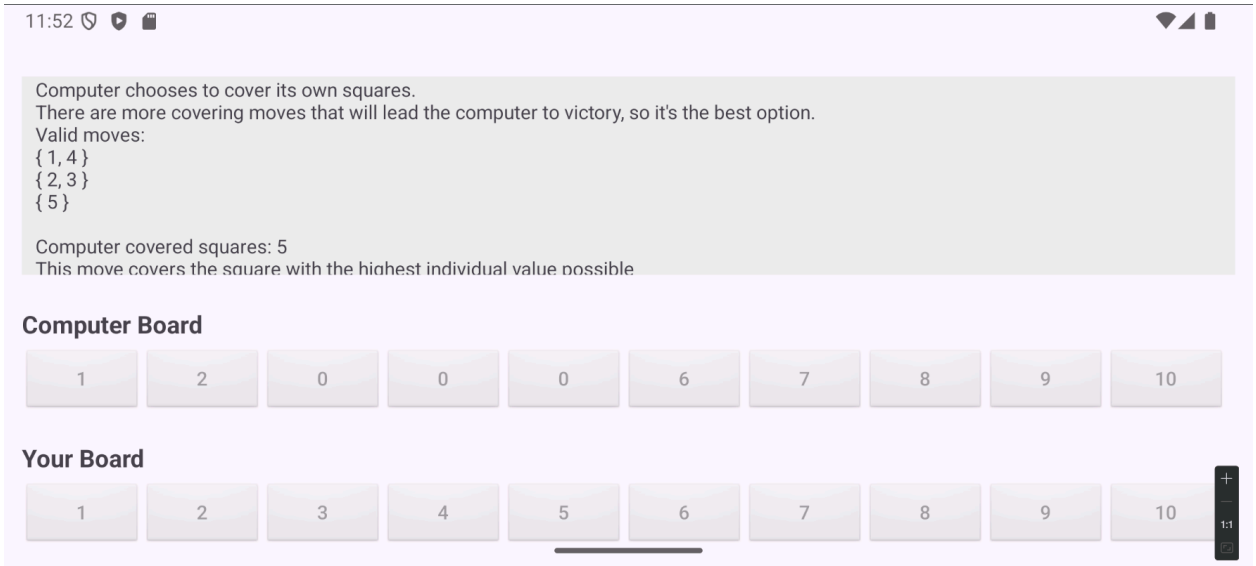
- The documentation function headers in my code are partially AI generated. I gave ChatGPT my functions and the format of the headers, and asked for it to complete the headers. I then edited them to my preference.

Screenshots

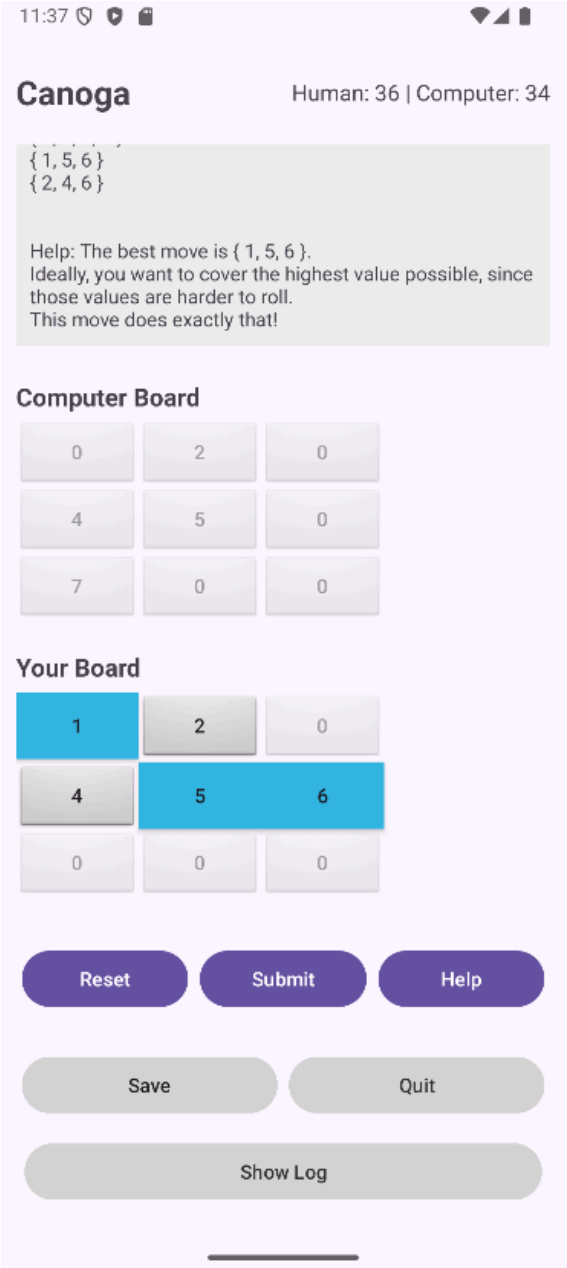
First Player of round being determined

Computer's move being explained

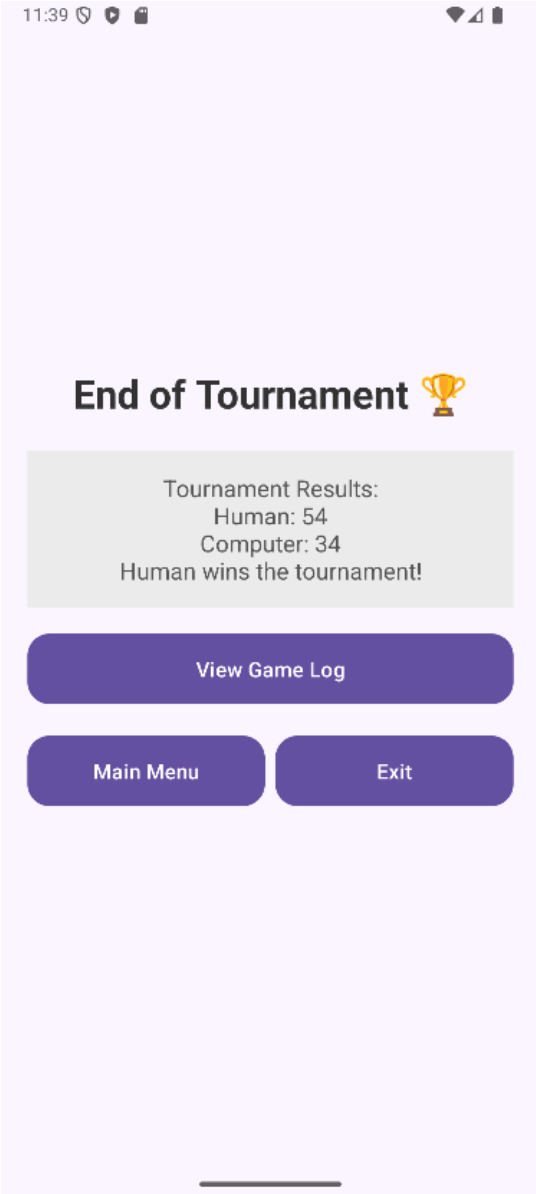



11:52
Computer chooses to cover its own squares.
There are more covering moves that will lead the computer to victory, so it's the best option.
Valid moves:
{ 1, 4 }
{ 2, 3 }
{ 5 }
Computer covered squares: 5
This move covers the square with the highest individual value possible
Computer Board
1 2 0 0 0 6 7 8 9 10
Your Board
1 2 3 4 5 6 7 8 9 10
1:1

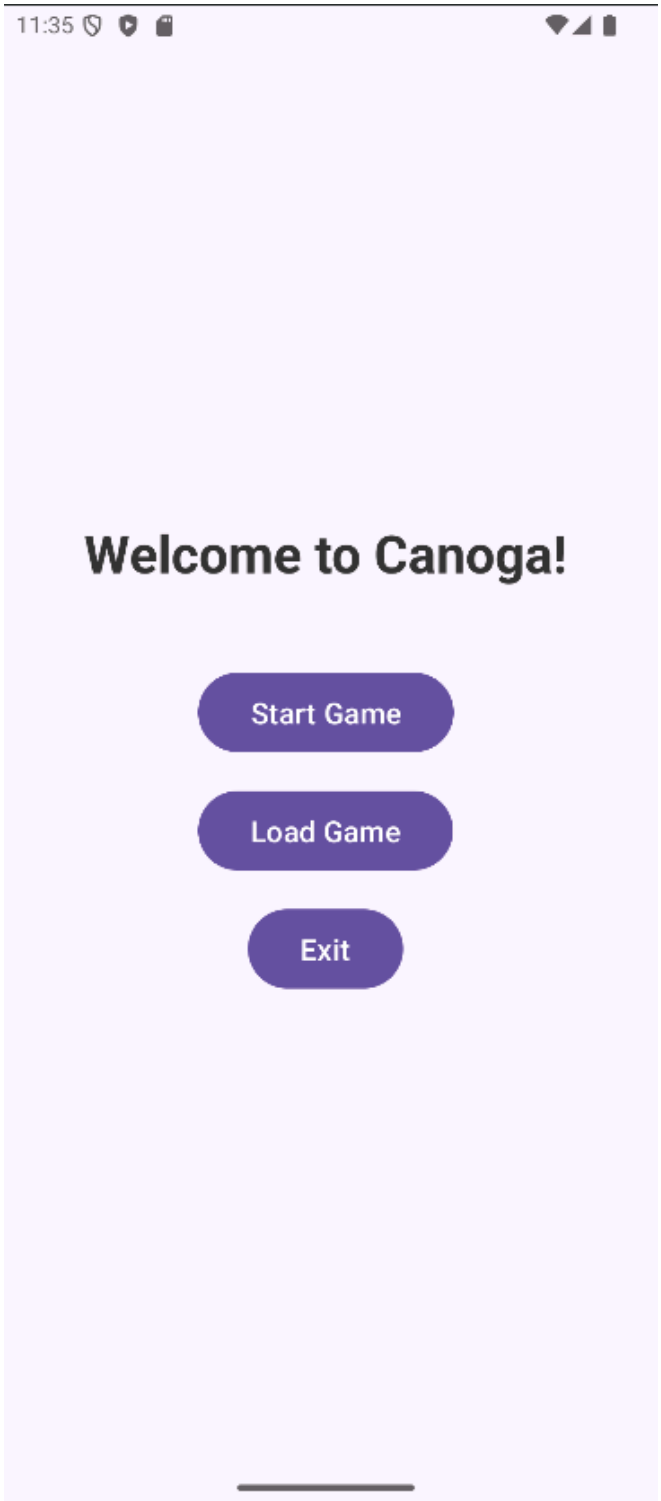
Computer providing help



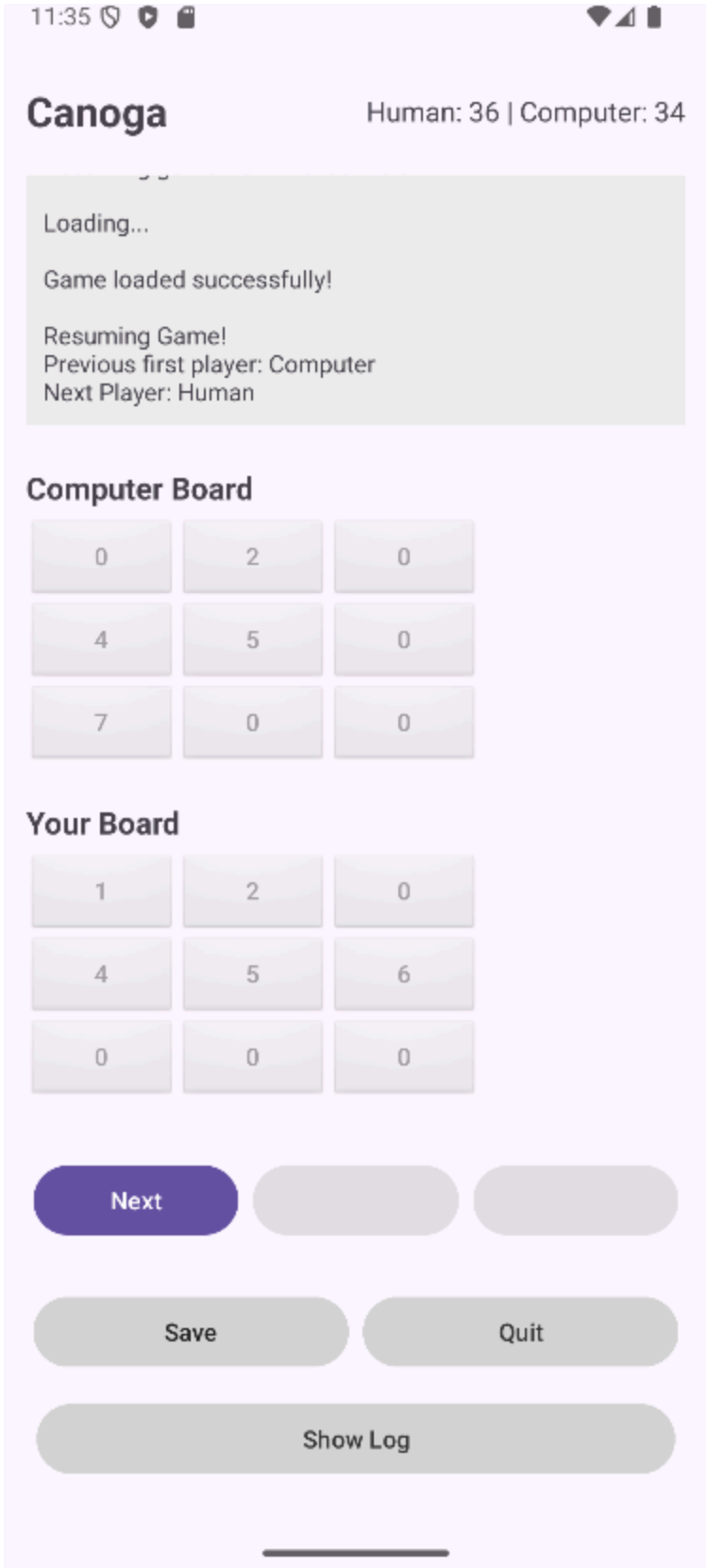
Winner of the tournament being announced

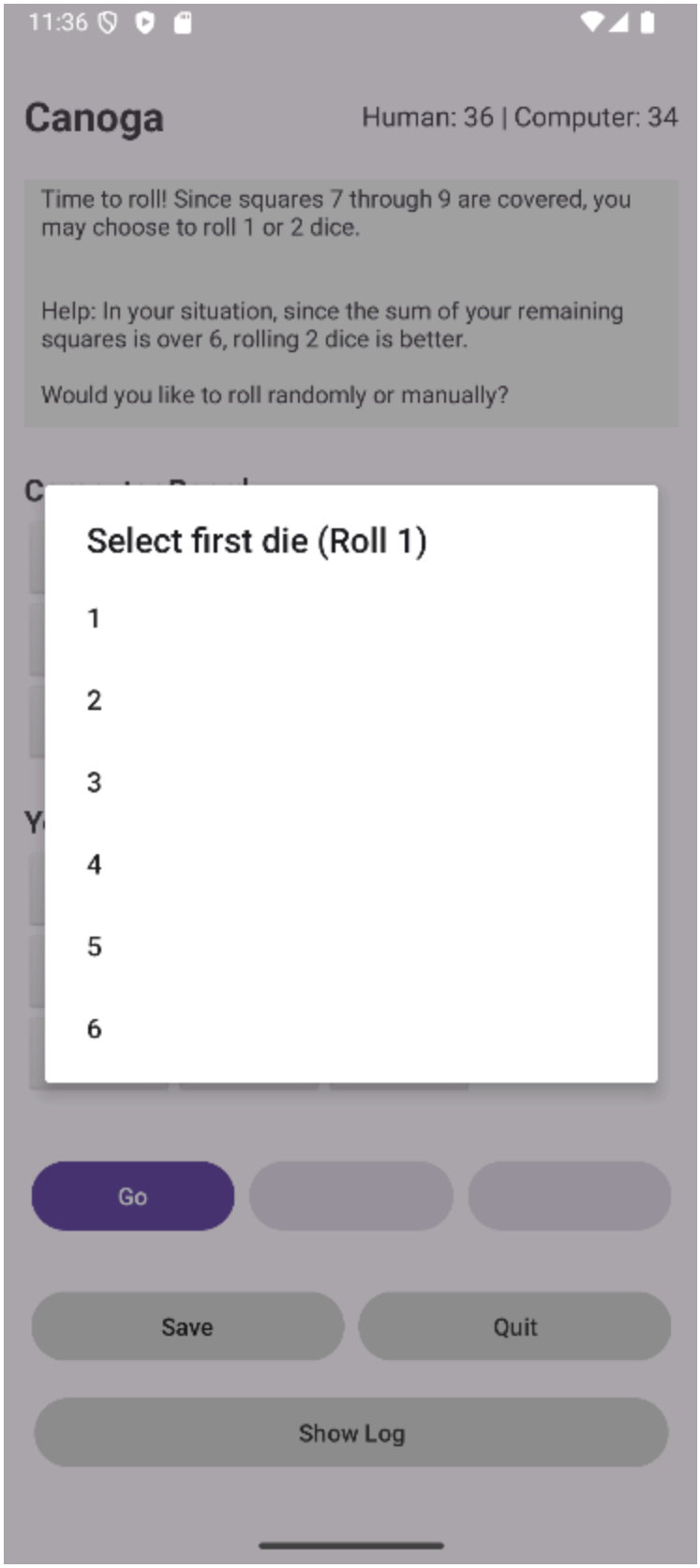


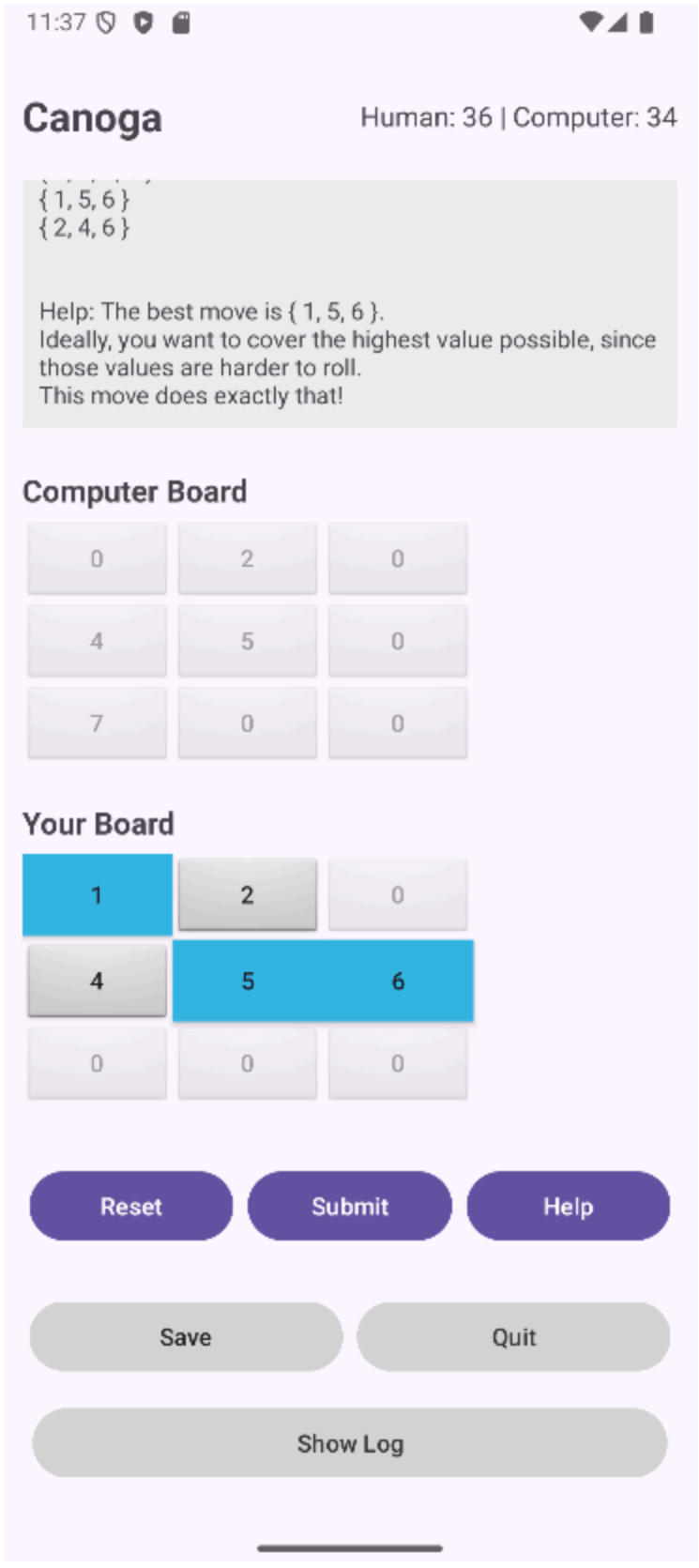
Playthrough of Serialization File 1 (Across the following screenshots)

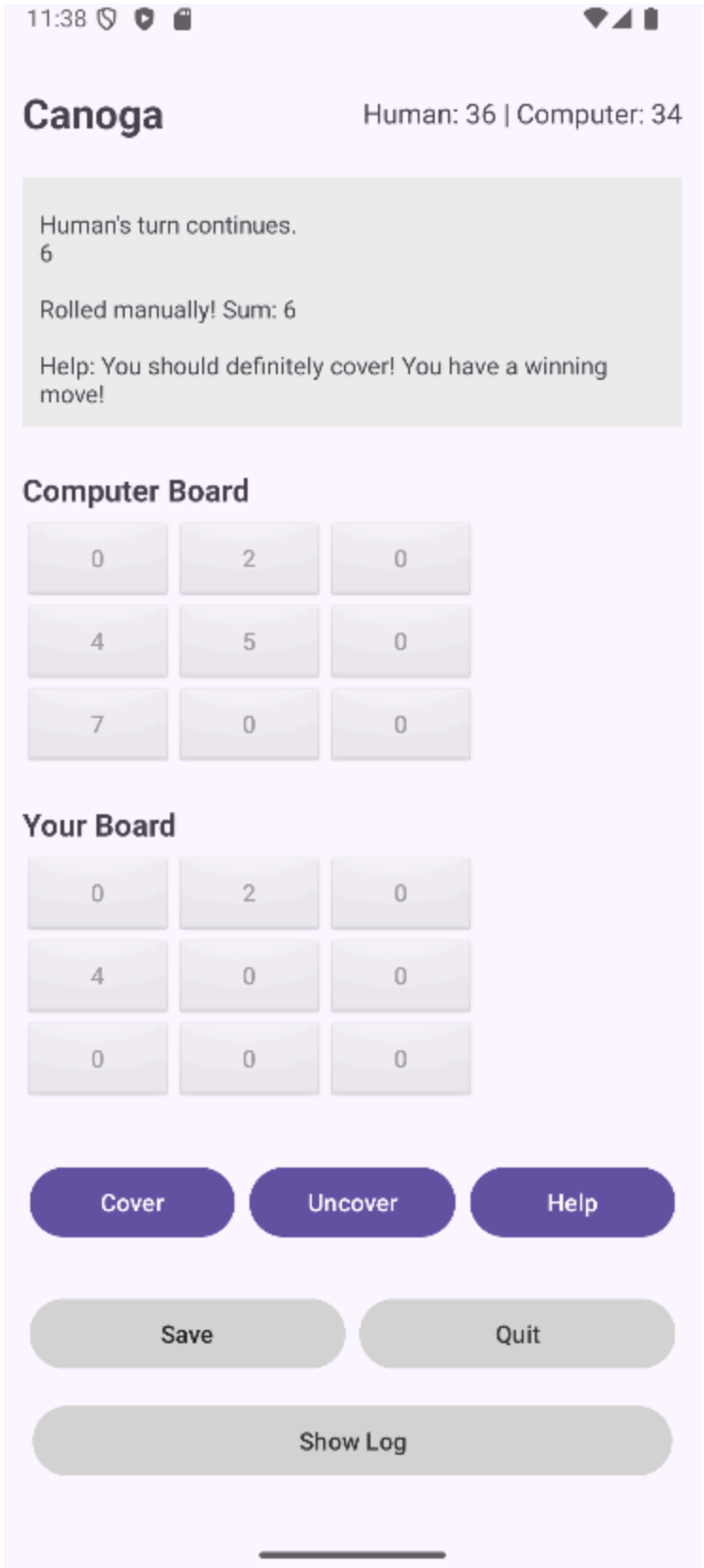


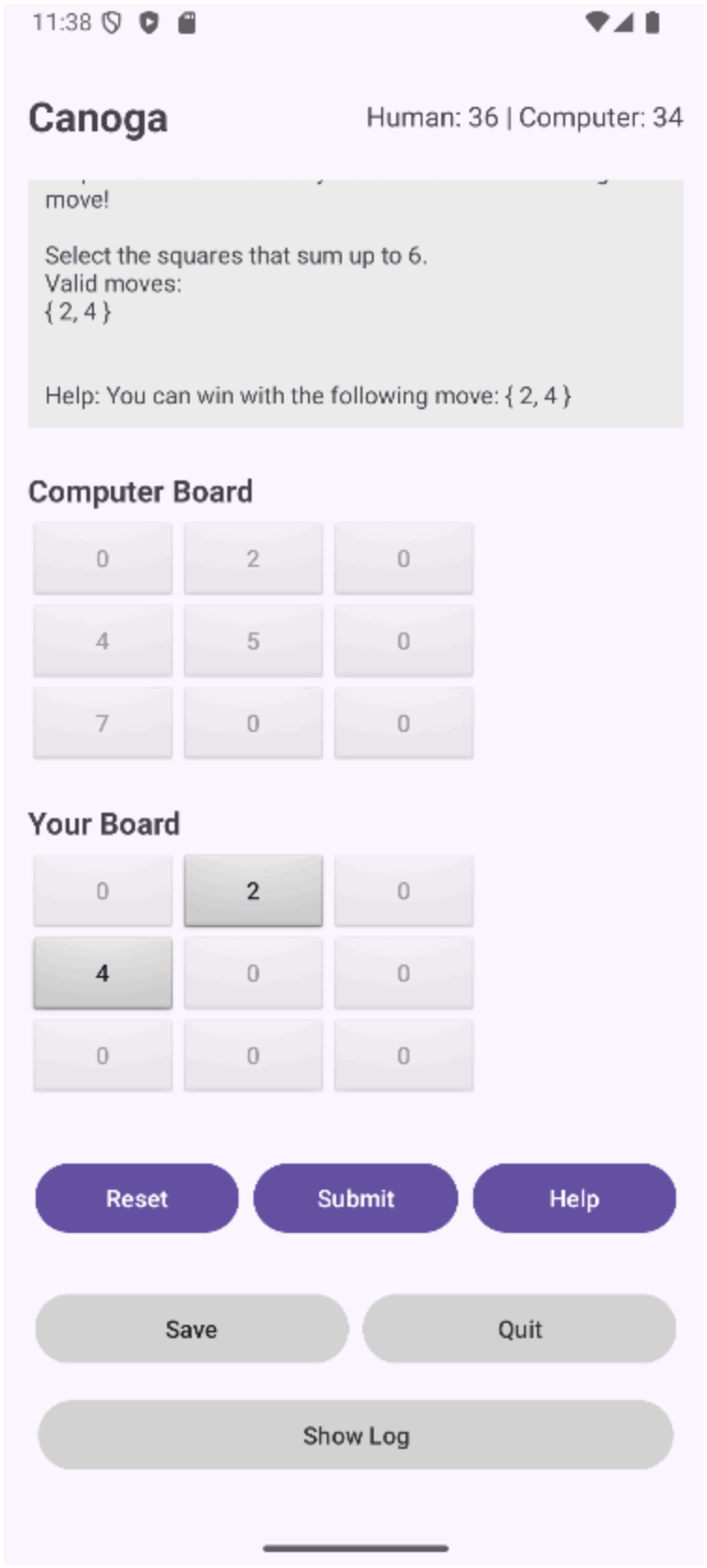

11:35
📂 Choose Saved Game
ser1.txt
ser2.txt
ser3.txt
ser4.txt
ser5.txt
game.txt
winWith10.txt
Back to Menu

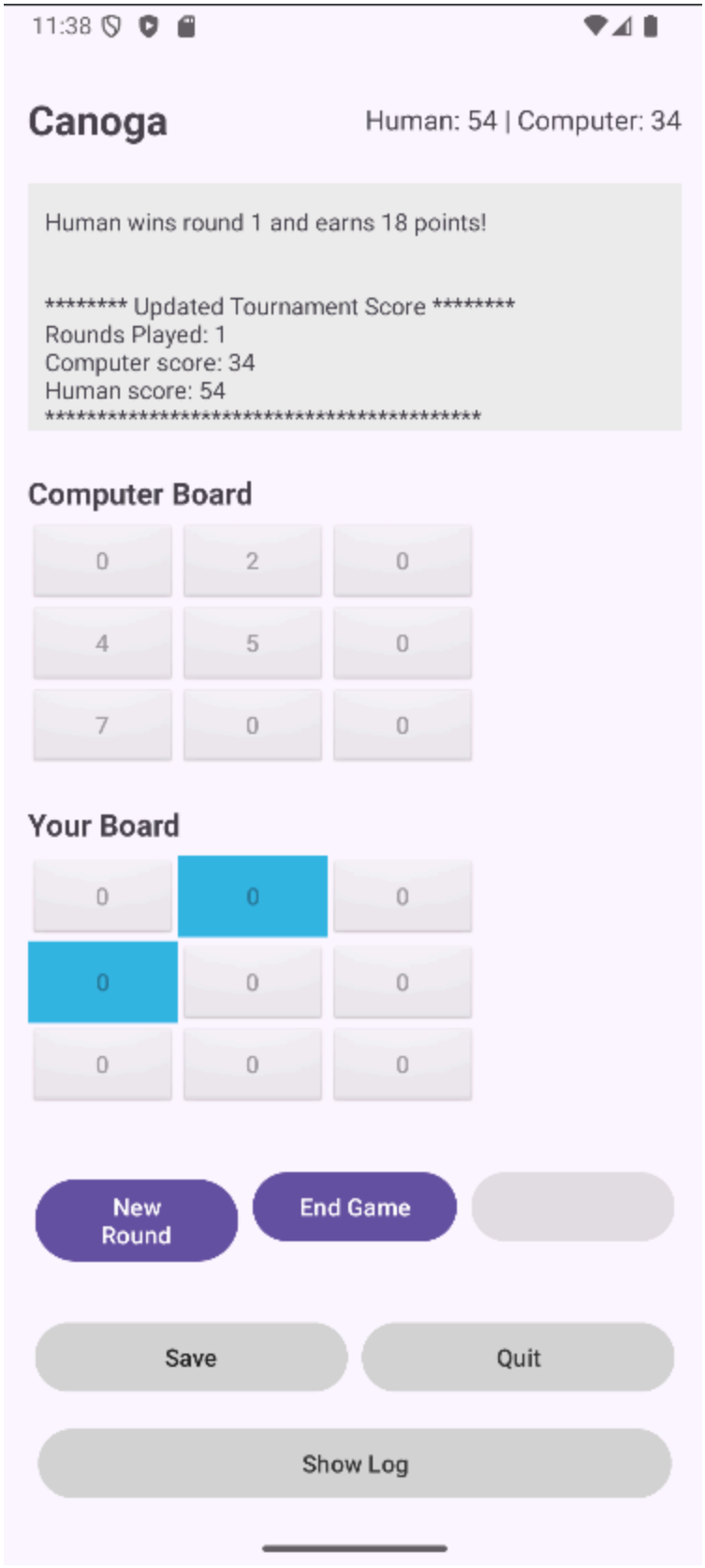

11:35
Canoga
Human: 36 | Computer: 34
Loading...
Game loaded successfully!
Resuming Game!
Previous first player: Computer
Next Player: Human
Computer Board
0 2 0
4 5 0
7 0 0
Your Board
1 2 0
4 5 6
0 0 0
Next
Save
Quit
Show Log

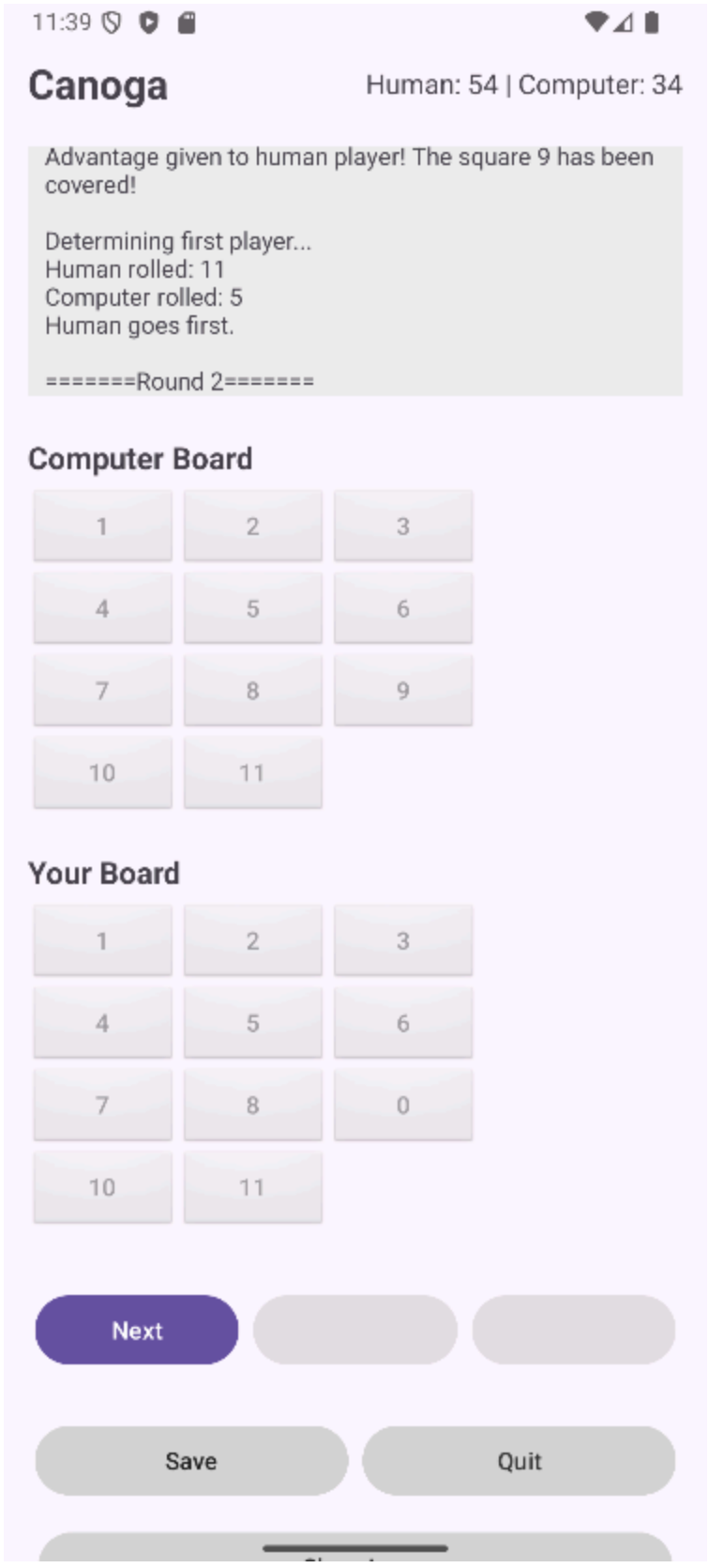

11:36
Canoga
Human: 36 | Computer: 34
Time to roll! Since squares 7 through 9 are covered, you may choose to roll 1 or 2 dice.
Help: In your situation, since the sum of your remaining squares is over 6, rolling 2 dice is better.
Would you like to roll randomly or manually?
Select first die (Roll 1)
1
2
3
4
5
6
Go
Save
Quit
Show Log

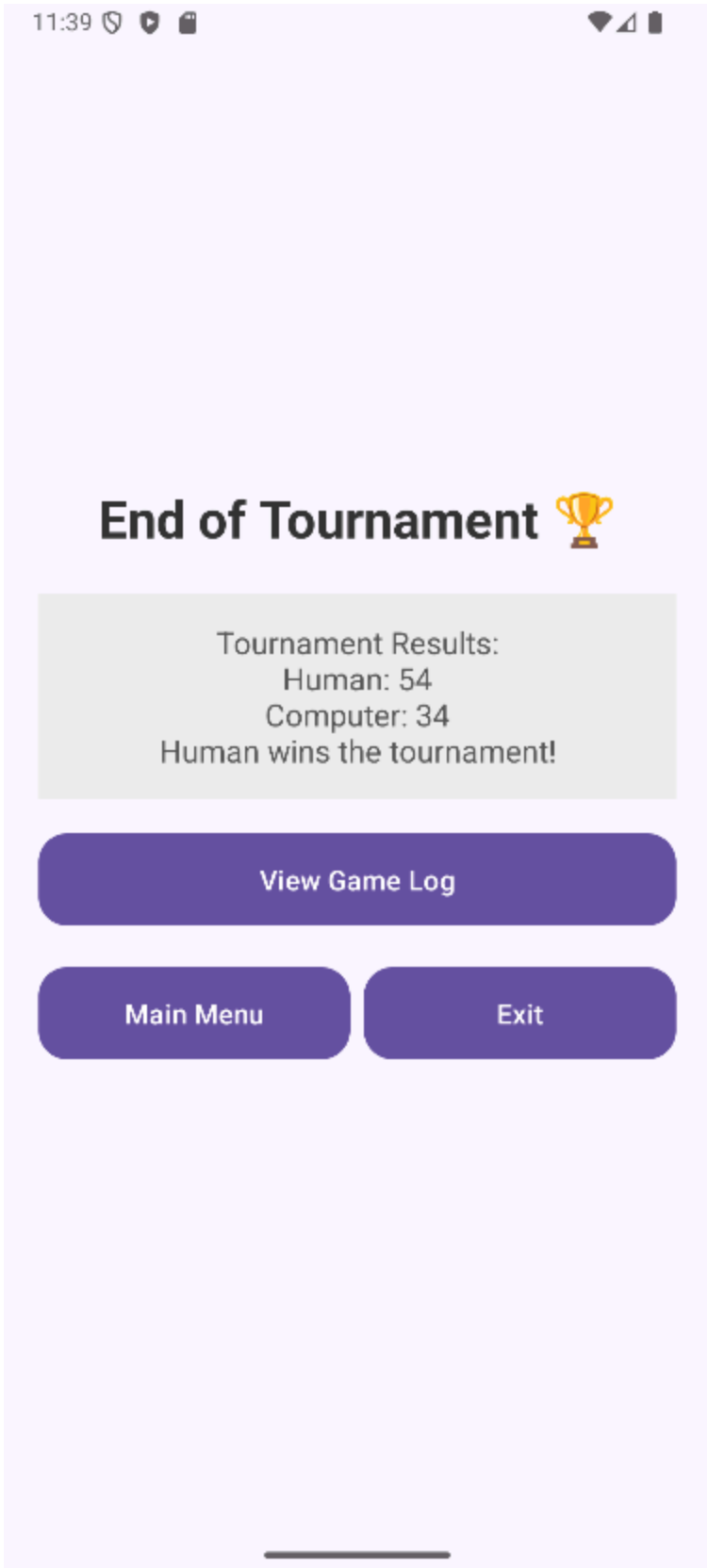

11:37
Canoga
Human: 36 | Computer: 34
{ 1, 5, 6 }
{ 2, 4, 6 }
Help: The best move is { 1, 5, 6 }.
Ideally, you want to cover the highest value possible, since those values are harder to roll.
This move does exactly that!
Computer Board
0 2 0
4 5 0
7 0 0
Your Board
1 2 0
4 5 6
0 0 0
Reset
Submit
Help
Save
Quit
Show Log

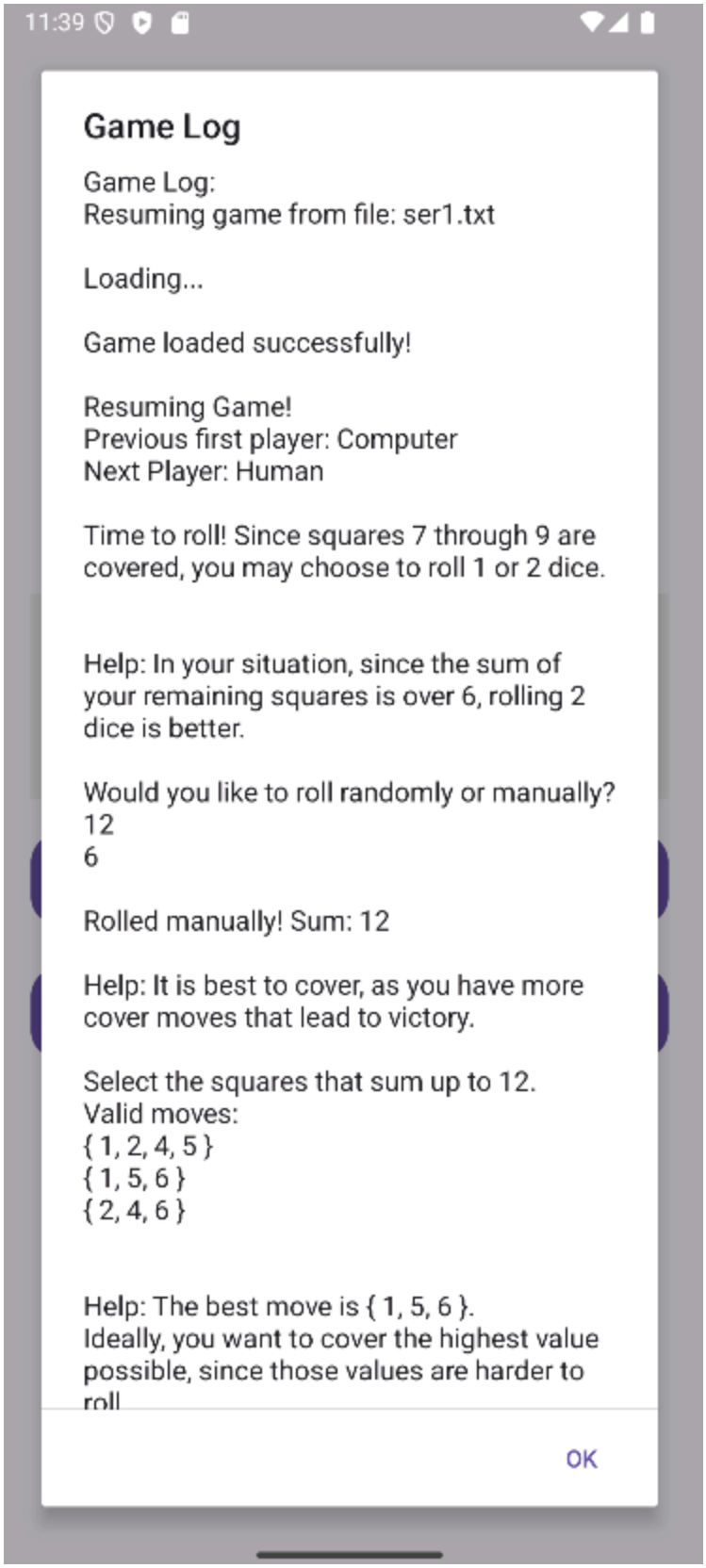

11:38
Canoga
Human: 36 | Computer: 34
Human's turn continues.
6
Rolled manually! Sum: 6
Help: You should definitely cover! You have a winning move!
Computer Board
0 2 0
4 5 0
7 0 0
Your Board
0 2 0
4 0 0
0 0 0
Cover
Uncover
Help
Save
Quit
Show Log


11:38
Canoga
Human: 36 | Computer: 34
move!
Select the squares that sum up to 6.
Valid moves:
{ 2, 4 }
Help: You can win with the following move: { 2, 4 }
Computer Board
0 2 0
4 5 0
7 0 0
Your Board
0 2 0
4 0 0
0 0 0
Reset
Submit
Help
Save
Quit
Show Log


11:38
Canoga
Human: 54 | Computer: 34
Human wins round 1 and earns 18 points!
******** Updated Tournament Score ********
Rounds Played: 1
Computer score: 34
Human score: 54
****
Computer Board
0 2 0
4 5 0
7 0 0
Your Board
0 0 0
0 0 0
0 0 0
New Round
End Game
Save
Quit
Show Log


11:39
Canoga
Human: 54 | Computer: 34
Advantage given to human player! The square 9 has been covered!
Determining first player...
Human rolled: 11
Computer rolled: 5
Human goes first.
=======Round 2=======
Computer Board
1
2
3
4
5
6
7
8
9
10
11
Your Board
1
2
3
4
5
6
7
8
0
10
11
Next
Save
Quit


11:39
End of Tournament 🏆
Tournament Results:
Human: 54
Computer: 34
Human wins the tournament!
View Game Log
Main Menu
Exit


11:39
Game Log
Game Log:
Resuming game from file: ser1.txt
Loading...
Game loaded successfully!
Resuming Game!
Previous first player: Computer
Next Player: Human
Time to roll! Since squares 7 through 9 are covered, you may choose to roll 1 or 2 dice.
Help: In your situation, since the sum of your remaining squares is over 6, rolling 2 dice is better.
Would you like to roll randomly or manually?
12
6
Rolled manually! Sum: 12
Help: It is best to cover, as you have more cover moves that lead to victory.
Select the squares that sum up to 12.
Valid moves:
{ 1, 2, 4, 5 }
{ 1, 5, 6 }
{ 2, 4, 6 }
Help: The best move is { 1, 5, 6 }.
Ideally, you want to cover the highest value possible, since those values are harder to roll
OK