Android Pente Manual

Bug Report:
To the best of my knowledge, there are no bugs within my implementation. I have extensively tested my program, but *unforeseen* bugs could exist - but again, to the best of my knowledge none exist.

Feature Report:
- Missing features: My computer does not delay winning for points; if sees that a move results in a win it will take it
- Extra features: For my views on the board, I put images on the pieces specifically for black and white stones which became black and white circles. I also added help functionality to display the position with a green circle to indicate to the user on where to place. Furthermore, I added X images for stones that could not be placed for a board restriction. Not entirely sure if these count as extra features, but they were not directly listed as needed within the rubric.

Help Received:
As most of the view and control were heavily researched on how to implement I felt it best to include a list of help received that extends generally over what I did for the view models. I will also include where I got the images from as to give credit to those who created them.

Coding Help:
- Intents: https://developer.android.com/guide/components/intents-filters
- Click Listeners: https://developer.android.com/reference/android/view/View.OnClickListener
- Visibility: https://developer.android.com/reference/android/opengl/Visibility
- MVC: https://www.geeksforgeeks.org/mvc-model-view-controller-architecture-pattern-in-android-with-example/
- MVC: https://academy.realm.io/posts/eric-maxwell-mvc-mvp-and-mvvm-on-android/
- Java Documentation: https://docs.oracle.com/javase/8/docs/
- Android: https://developer.android.com/guide
- Eight Puzzle from class
- A few questions or issues were handled by the editor itself, suggesting syntax warnings. I've tried to mention where to the best of my ability within the code

Images for Views:

- All circle images: https://pngimg.com/search_image/?search_image=circle&search_button=Search
- X for board restriction: https://commons.wikimedia.org/wiki/File:Red_X.svg
- Quarter for coinflip: https://www.vexels.com/merch/png/half-dollar/

Description of Data Structures/Classes:
Model Classes: Board, Codes, Player, Human, Computer, Round, and Serialize

View/Control Activities: LaunchActivity, CoinTossActivity, FileSelectAcitivity, MainActivity, SerializeActivity, AnotherRoundActivity, and EndTournamentActivity

Notes:

- ***IMPORTANT:*** *the model is completely separate from the control and views. The controller calls the model, and gets a value in return. This value is then given to the view functions to display.*
- *Each activity has a corresponding xml file in the layouts for the specific view*
- *The view and control activities/classes are combined. Meaning control and view share the same class but are distinctly mentioned within the code. I separated views into different functions that control functions to create a clear separation between the two.*

Model Classes:

Class Board:
Description:

Board handles all the logic of the game board. This includes placing a stone on an intersection, checking for 5 in a row stones, checking and updating if there is a capture pair. It is responsible for handling the state of the Pente game board.

Inheritance:

No inheritance used for this class.

Interfaces:

Implements Serializable - for copying with intents, and Cloneable to create a copy of the object

Composition/Data Structures:

- Uses a 2D vector of characters to represent the board itself. Uses a vector instead of array to allow for multiple board sizes - not included in current implementation but much easier to modify at a later time
- Has a move ‘struct’ (really just a nested class) to store all the information about a move. Includes information like the position, the intersections left, and the stones

around position before the board was updated. Created to facilitate undoing moves

- Has a PositionPair ‘struct’ (really just a nested class) to be able to update and pass primitive types of row and column in my ParsePosition method. Makes it easier instead of multiple function calls for what in reality is one step.
- A stack of move structs that represent the previous moves made. It is used to be able to undo moves (helpful for the computers strategy)

<u>Class Codes:</u>

Description:

Handles the return codes and error handling that may occur anywhere throughout the program. It stores the error as an enum and has a member that allows you to get the error message associated with it.

Inheritance

No inheritance used for this class.

Composition/Data Structures:

Uses an enum to identify the return code. Works well for smaller projects.

<u>Class Player:</u>

Description:

The Player class serves as the base class for Pente players. It provides common functionality and attributes that both Human and Computer players share. The class holds player-related data, including the player’s name, color, scores, and the best move to make in the case of a Computer player.

Inheritance:

The Player class doesn't inherit from any other class. It serves as the base class for the Human and Computer players. It defines a pure virtual function MakeMove, which must be implemented by derived classes to make a move on the game board. And provides two protected members BestMove() and GetReasonMessage() for Human and Computer to use to get the strategy. It also placed two functions EvaluateMove() and DetermineBest() in protected that can be overridden by a base class to change how the strategy evaluates the move or determines the best one - strictly for modifiability at a later date. It also has a GetHelp() method to get the best position if asked for.

Interfaces:

Implements Serializable - for copying with intents, and Cloneable to create a copy of the object

Composition/Data Structures:

- Member variables such as m_name, m_color, m_tournamentScore, m_capturedPairs, etc., are used to store player-related data

- Has a struct called ComputerMove, which is used for the computer strategy, is composed within the class to store information about the best move to make
- m_bestMove is a protected member that is a ComputerMove struct that the derived classes can access
- It also contains MoveReason enum that identities why the computer played where it did

<u>Class Human:</u>
Description:

The Human class represents a human player in the game of Pente. It is a subclass of the Player class, inheriting common player attributes and functionality. The Human class is responsible for making moves on the game board based on human input and interacting with the game.

Inheritance:

The Human class inherits from the Player class, which provides the basic player-related attributes and methods. It specifically overrides MakeMove.

Interfaces:

Implements the same as Player: Serializable - for copying with intents, and Cloneable to create a copy of the object

Composition/Data Structures:

- It implements the MakeMove method, which is a required override from the base Player class, and is used to make a move on the game board

<u>Class Computer:</u>
Description:

The Computer class represents a computer player in the game of Pente. It inherits from the Player class. The Computer class is responsible for playing the best possible move.

Inheritance:

Inherits from the Player class, which provides common player attributes and methods. By inheriting from the Player class, the Computer class can use the BestMove protected member to make its move on the board.

Interfaces:

Implements the same as Player: Serializable - for copying with intents, and Cloneable to create a copy of the object

Composition/Data Structures:

- It implements the MakeMove method, which is an override from the base Player class. Basically it just places the stone based on the BestMove

<u>Class Round:</u>

Description:

The Round class represents a round of the game Pente. It manages the gameplay between two players, a human player and a computer player. The class keeps track of the game state, including the game board and player turns, checking for a round's end and tallying scores.

Inheritance:

The Round class doesn't inherit from any other class. It is a standalone class.

Composition/Data Structures:

- Has two player objects, a Human and a Computer, as member variables. This is needed as we need to know the human to perform a coinflip.
- It uses a player vector, m_players, to store the two players in the order of their turns
- The class includes member Board object which represents the game state
- Other member variables are used to manage the game, including the current player index, the winner of the game, etc.

View/Controller Activities:

LaunchActivity:

Description:

The LaunchActivity is where the user is first brought to when the app is open. It consists of a title/splash screen welcoming the user to the game and has two buttons to resume and start a new game from. Acts as the control for the overall tournament, this is where the main round model is.

Inheritance:

Extends the Activity class in Android.

Control Composition:

- Consists of two input buttons, Resume and NewGame.
- Resume will get the available files through the model, package them into an intent and go to FileSelectAcitivity to resume the game from a file
- NewGame will get the round object and the CoinTossAcitivity with be started

View Composition:

- XML file can be found under res/layouts/activity_launch
- Displays a message if there were no files found within the saves folder via DisplayNoFiles()

CoinTossActivity:

Description:

This is where a game goes if it is brand new or if the player wanted to play another round and the scores were tied. Handles the input for users' decision on which side of the coin they want to use. Can be started by LaunchActivity or AnotherRoundActivity.

Inheritance:

Extends the Activity class in Android.

Control Composition:

- Control gets the round model from an intent and uses said model to perform a coin flip based on user input.
- There are 3 buttons, a coin to flip it (a picture of a big coin), a toggle button that displays heads or tails when pressed, and a continue button for when after the quarter/coin flip button is pressed.
- Continue button is displayed after the result and starts the MainActivity.

View Composition:

- XML file can be found under res/layouts/activity_coin_toss.
- Text view header to display coin toss, a quarter image for the bottom, and a hidden text view for the message if the user won or not.
- There is a hidden text view that is set to the winner and who goes first after the coin flip button is pressed. Simply displays the winner based on what it received from the control.

<u>FileSelectAcitivity:</u>

Description:

Where the user can select a save from. Includes a drop down menu for each file that ends with .txt.

Inheritance:

Extends the Activity class in Android.

Control Composition:

- Has a drop down menu and a continue button.
- The drop down menu displays the files, and the continue button starts the MainActivity based on the selected file.

View Composition:

- XML file can be found under activity_file_select
- There is a header saying to choose the file, the drop down menu, and the continue button.

<u>MainActivity:</u>

Description:

The bulk of where the user will be while playing Pente. Can be started by FileSelectAcitivty or CoinTossActivity. Handles the user input, the game log, the board display, the scores, the help and the win. As well as ways to quit or play another round

Inheritance:

Extends the Activity class in Android.

Control Composition:

- There are two major components to the control, the board and the user controls at the bottom of the screen which contain three buttons. Of course the controls only call the model - the actual logic does not happen here.
- For the board, it is a 19x19 grid of buttons, each with a tag that represents which position it is. When it is the humans turn they can click on it and the control gets that position and places it on the model board.
- The user controls at the bottom has a step button, for when it's the computer's turn. This is to allow the computer not to go too quickly and basically make it up to the user how fast the computer responds.
- There is a help button in user control that calls the model to get the best position, this is displayed on the board via a green stone.
- There is a save and quit button which takes you to a new activity to save the current game.
- There is also an invisible container on the right side of the activity that appears when the game is ended. The quit takes you to EndTournamentActivity and the play another round takes you to the AnotherRoundActivity.

View Composition:

- XML file can be found under res/layouts/activity_main
- The screen is divided into four main parts, the game log, board, user control and the right side which displays the current scores, the next round/quit buttons for when the game ends and the help when it is called.
- The left side is the game log, all messages are appended (got from the controller) to the text view on a scrollbar element. This allowed the user to see the log of the game
- The middle is where the board is, which is dynamically rendered/generated for the buttons. If a stone is black, it uses a black image, white image for white an X if there is a board restriction, and a green stone for when help is called. No model is in here, it gets the vector of the board to display it.
- The user control as stated in the control composition has three buttons and a text view in the middle to display the current player. The help is disabled when it's the computer turn, the step is disabled when it's the human turn and help can only be clicked once before it is not clickable anymore.
- The top right side is where the scores for captured pairs and tournaments are.
- When the help button is clicked, which the controller tells it to, it makes a text view visible with where to go in English and sets the stone position button to green.

- The bottom right is where two buttons are displayed, quit or play another round - only displayed when the controller informs that the round is ended.

<u>SerializeActivity:</u>
Description:

The place where a user can go to serialize a game state. Can be started by MainActivity. It allows for user input on the file name specifically. Basically a way to save the game.

Inheritance:

Extends the Activity class in Android.

Control Composition:
- Three buttons are present: cancel, quit and save.
- Cancel takes you back to the main activity in case you did mean to press quit/cancel.
- Quit ends the game and takes you back to the launch activity.
- Save takes user input via an input text viewer and saves the game state, taking you back to the launch activity if successful.

View Composition:
- XML file can be found under res/layouts/acitivty_serialize.
- Text view that states to save a game, and the three buttons.
- A message appears in a text view if the model (told to this view by the controller) couldn’t save.

<u>AnotherRoundActivity:</u>
Description:

This activity displays to the user to play another round or not. Serves as the middle-man between starting a new activity from the previous round. Can be started by MainActivity - only when the game ends.

Inheritance:

Extends the Activity class in Android.

Control Composition:
- Has only 1 button which is to go to the next round, labeled continue.
- From the model the controller knows if the tournament scores are tied or if one player has more than the other. If tied it will start the CoinToss activity after it tells the round model to reset, if there is a person ahead it will take them directly to the MainActivity.

View Composition:
- XML file can be found under res/layout/activity_another_round.

- Has a textview as a header to display go to another round, a textview for the current scores, and a textview to display the message if the new round needs a coin flip or if a user is going first.

<u>EndTournamentActivity:</u>

Description:

This is the activity for when the user wants to end the tournament after the round ends. Can be called by MainActivity, and its sole purpose is to display the tournament scores and the winner of said tournament.

Inheritance:

Extends the Activity class in Android.

Control Composition:

- Only 1 button, which is quit, which starts the LaunchActivity to officially end the tournament.

View Composition:

- XML file can be found under res/layouts/activity_end_tournament
- Has a text view header to display who won (given to by controller) and the scores of the tournament. And of course the button listed in control.

Log:

Nov 7th 2023:

- Created a new Android empty app, and learned some of the workings of Android Studio (30 mins)

- Noted and reviewed Vectors, Arrays, Stacks, Classes and how Objects are references in java. Before starting any work on the actual game itself, I needed to learn how things would interact to ensure my code is reliable, safe and ensures encapsulation. Specifically with how Objects are treated as references (1.5 hours).

- Ported over some Board.h and some of Board.cpp from the C++ project into Board.java. Almost everything will have the same structure as the C++ project.
- Rewrote the class constants for the Board. Most notably changed the ROW_DELTA and COLUMN_DELTA to be a proper Java array. Meaning no set size as the compiler figures that out for us.
- Made the Move struct into a private class - since no structs in java - for board to facilitate storing moves. In essence, like the C++ project, it stores the position, bounds, wins, intersections left, captured pairs and the previous sequences from the last move made.
- Update the member variables to work in java. Specifically m_gameBoard, which is a 2D Vector of characters holding the ‘stones,’ the m_prevMoves which is a stack that

stores all the moves made on the board, and m_currMove which stores the current move. All of these are initialized in the constructor for Board.
(1.5 hours for the above 4 items)

- Starting moving some of the accessors and public utility members over from the C++ project (30 min).

- Added TODO comments in places of what I need to do next. Specifically with porting over code from C++. Most importantly adding direction in porting over my Codes class in an efficient and proper way that is clean and easily modifiable (30 mins).

- For the future: I still need to add a bunch of the functionality for private member functions and a BoardView class to test if my codes are working.

Total 4.5 hours

Nov 28th 2023:
- Ported over almost all of Board.cpp into the Board class. Updated functionality to match Java syntax and ensured encapsulation of data members (2 hours).
- Updated Board.InitGameBoard() to properly return a vector of characters calling it within the Board constructor (30 mins).

Total 2.5 hours

Nov 29th 2023:
- Finished porting board entirely, all functions and member variables are completely consistent with java syntax (2 hours).
- Updated how Board.ParsePosition() in C++ used pass by reference to get the rows and columns as integers. Included a PositionPair to essentially as a struct of row and column to be able to pass them by reference and make things easier (30 min).

- Extensively tested board class, ensuring all functions that worked in C++ are now working in java (1 hour).

3.5 hours

Nov 30th 2023:

- Updated Board.PotionalCaptures which was having issues in my previous testing session. Tested and now works properly in conjunction with the Player class (1 hour).

- Ported over Player, Human and Computer classes over from C++. All methods appear to be working.
- Made the struct for ComputerMove a nested class to essentially replace the struct. It will functionally serve as a struct for my functionality.
(2 hours for the above 2 items)

- Researched how to override the clone method to ensure proper copies of my board instead of a shallow copy. Especially important in Board and Player classes
- Created clone() methods for Board and the Move nested class inside of it.
- Created clone() methods for Player and the ComputerMove nested class inside of it.
(2 for the above 3 items.

Total 5 hours

Dec 1st 2023:
- Tested Player class along with Human and Computer classes to ensure proeply functionality from the C++ port. Everything appears to be working as expected (45 min).

- Ported Codes class and the enum from C++, with a bit of research in the java docs to see how enums can be used in java (15 min).

- Ported over Round class from C++ and all the functionality it needed (2 hours)

- Ported over Tournament class from C++
- Did extensive research on how to save files in java, specifically with the java.nio Path and File systems to try and replicate how files work on Java and how to use them
- Implemented this knowledge to read and save files
(2 hours for the above 3 items)

- All model classes were successfully ported over, and Pente was successfully working on the commandline. Did more testing to ensure it was working as intended (30 min).

- Began starting the android project doing extension research on how activities work, and how to use intents (1 hour)

- Create the launch activity and the coin toss activity. Both with very limited functionality. I also spent some time learning how to use the design feature in android studio to place views and such in the proper places (2 hours).

Total 8.5 hours

Dec 2nd 2023:

- Stripped and got rid of the tournament model class, which in place I created a Serialize class to implement the saving and reading from files after realizing I probably would not need my tournament class/model (10 min).
- Included all the saving implementations that were essentially generalized from my when porting over from C++, such as formatting board players and next player, as well as parsing them in a line by line format, but adapted it to the assets folder in android (2 hours)
- Read up on how to use assets folders in android to read from files (1 hour)
- Created a FileSelectActivity to read from said files in an elegant manner. (1 hour)

- Updated my coin toss and launch activity within the XML file and the android studio design section. Spent time understanding how buttons work and how to use text views to display information (1 hour).
- Updated launch activity with buttons for a new game and resume game, which were event listeners for the coin toss activity and file select activities (1 hour).

- Created my main activity class to house the board, future game log, scores and a user controllable panel (1 hour).
- Researched and experimented with multiple formats until asking Alan Salzano on which layout he used, which he exclaimed was a combination of two linear layouts, for the row and column headers, and a frame layout to actually store the buttons. Most of the time was spent testing this layout to ensure it fit my needs (2 hours).
- Created a generator for the board buttons, which basically made their tag and size, once again suggested by Alan, the position of where they are in order to have proper event listeners (1.5 horus)
- Added images if the stone was black, white or had a bounds restriction to make it more intuitive for the user (30 min).
- Made a required input member within the player to disable certain buttons in the user control panel and on the board so that it made it more clear where a user can and can't go (20 min).
- Tested my board completely to ensure that the GUI was working properly.

- Updated the FileSelectActivty to have a drop down from the available saves, which my model helpfully provides in the Seralize.ReadAvailableFiles(), which I just made (30 min).

- Updated my Round class to remove the Round.Play() method and its while loop to actually be event based - solely using Round.FacilitatePly() now (30 min)

- Created a GameLog class with a static vector of strings, and replaced all my System.out.println() with it that was included in the command line version. Made it super simple to use (1 hour).
- Create a text view and scroller on the xml page, for this new game log (30 min)
- Created a view function to be able to append messages to this textview and scroller (30 min).

- Went to bed.

Total 14.5 hours

Dec 3rd 2023:
- Read up a bunch on MVC to ensure what I had been doing was correct, realize some code in the view controller should really be in the model and updated it accordingly (2 hours)

- Created EndTournamentActivity to display winner results and the ending scores for the tournament. Called this within the main activity once the game was over and was called from a button which was made visible at the end of the game. Displays winner and scores and a button back to the launch activity (1 hour)

- Created SerializeActivity to save a file based on user input, and then realized that you could not save to the assets folder in android - I discovered a massive bug, and spent quite a bit of time trying to figure out what was wrong (2 hours).
- Understood that I know need to use local storage, I rewrote how to do my Serialize class to encompass the local files on the android emulator/device (2 hours)

- Created AnotherRoundActivity to allow users to play another round once their one was finished, and include the view within the XML file (1 hour).

- Move my GetHelp function into my player class to make it truly polymorphic and appear isomorphic. This included my have to set input to be less confusing when reading
- Added the help viewer on the right side based on an array of string, [0] for the position [1] for the reason (not a great idea but tired) and made the position highlight green on the board itself.
(2 hours for the above 2 items)

- Reviewed and revised a bunch of activity classes including: another round, coin toss, serialize and main. Tried to make the layouts as intuitive as possible and extended some features, as well as some colors, to make everything more appealing (2 hours).

- Removed all TODO comments and updated a bunch of comments within my model and views to try to better explain what my code was doing (1 hour)

- Extensively tested my GUI to ensure it was working properly, fixed all the bugs that I found - but was mostly solid as I had been testing regularly after each activity. Ready for DEMO (1.5 hours)

- Went to bed at 6am.

Total 14.5 hours

Dec 4th 2023:
- After the project demo, I updated my Player.GetHelp() method to return the ComputerMove() 'struct' instead of an array of strings. Much better format, allows me to get the position and the message in a more clear manner (30 min).

- Added proper headers to all of my model classes. Which includes Board, Round, Player, Human, Computer, Serialize, GameLog and Codes (1.5 hours).
- Added function headers to all my view classes, and updated any comments that were unclear. Specifically adding more where I thought were necessary and removed comments that were not clear (1.5 hours).

- Created my manual for the project, including bug report, feature report, a note on help received, and began working on my description for my model classes (1.5 hours).

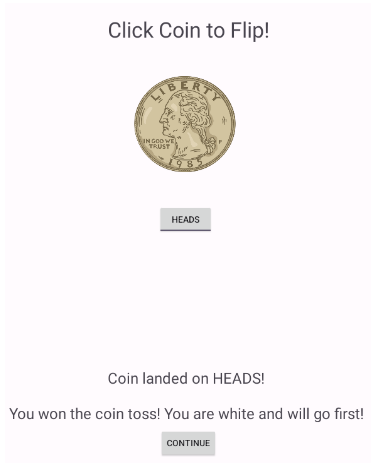
Total 5 hours

Dec 5th 2023:
- Tooks screenshots of the game and put them onto my manual (15 min)
- Created the class description and data structures for my view and controller classes/activities within my manual (1.5 hours).

Total 2.5 hours

How to Run:
Firstly, get your hands on an android device or emulator, and more importantly Android Studio. **IT IS HIGHLY RECOMMEND THAT YOU USE A LARGER SCREEN DEVICE AS SMALLER SCREENS WILL SQUEEZE THE DISPLAY!** The AndroidManifest.xml will have the launch activity listed, if not the file is LaunchActivity - which is essentially the entry point. Hit the play button and the emulator of your choice will launch, and you can select a new game from there.

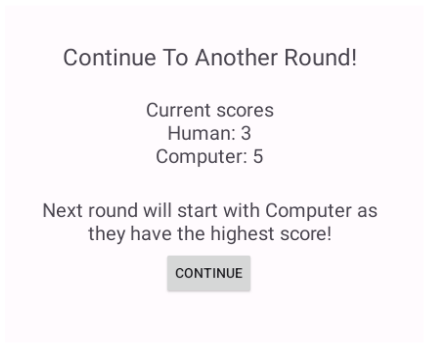
Screenshots:
<u>First player of round being determined</u>
Tied scores or first round:


Click Coin to Flip!

Coin landed on HEADS!

You won the coin toss! You are white and will go first!

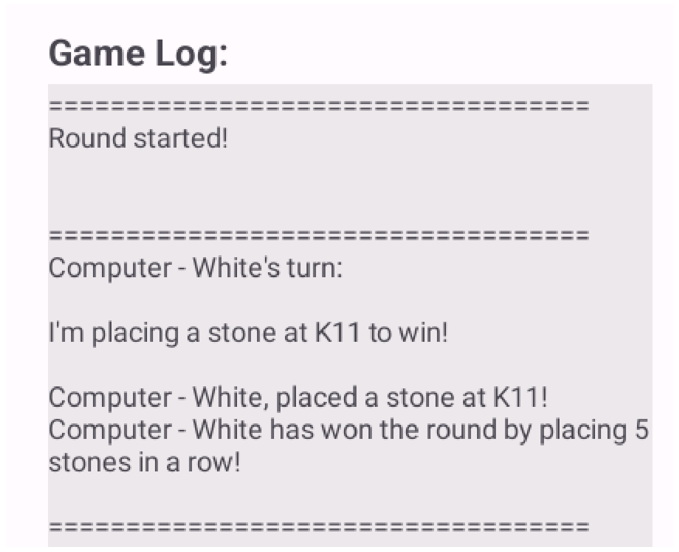
One player has more points:


Continue To Another Round!

Current scores
Human: 3
Computer: 5

Next round will start with Computer as they have the highest score!

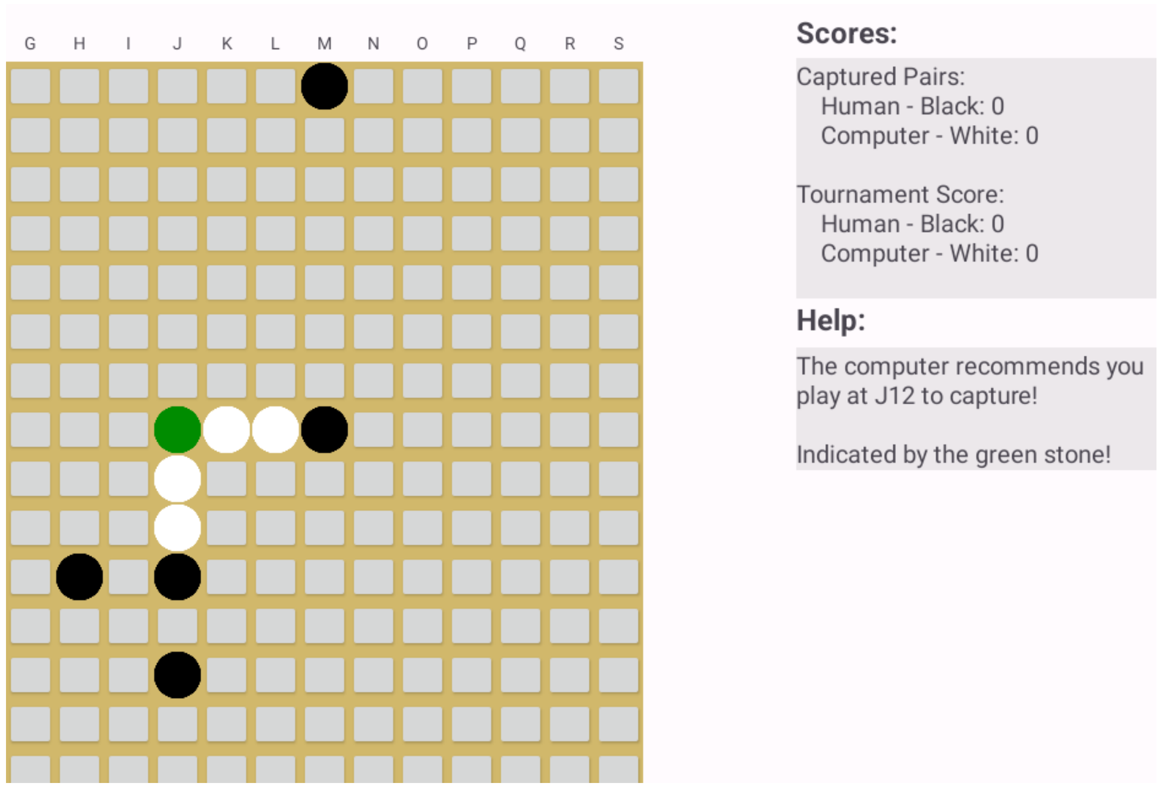
Computer's move being explained



Game Log:

```
====================================
Round started!


====================================
Computer - White's turn:

I'm placing a stone at K11 to win!

Computer - White, placed a stone at K11!
Computer - White has won the round by placing 5
stones in a row!

====================================
```

Computer providing help



Scores:

Captured Pairs:
Human - Black: 0
Computer - White: 0

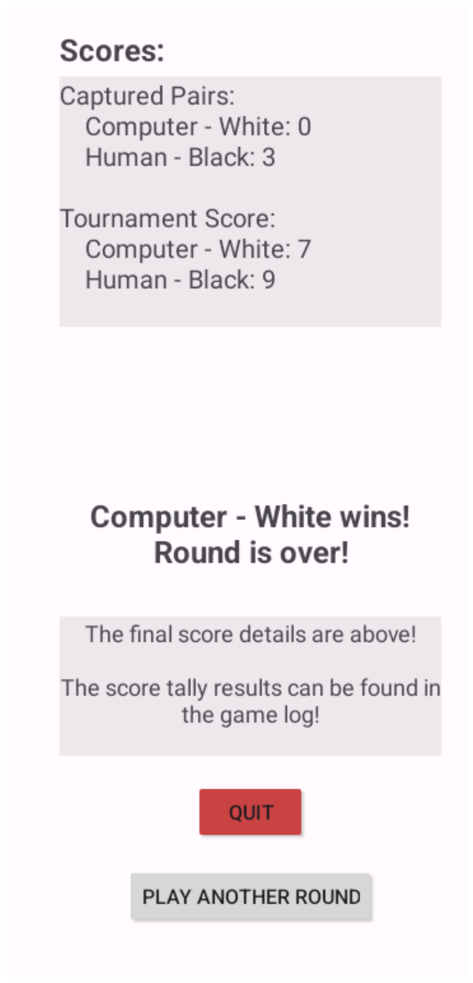
Tournament Score:
Human - Black: 0
Computer - White: 0

Help:

The computer recommends you play at J12 to capture!

Indicated by the green stone!

Winner of the round being announced



Scores:

Captured Pairs:
Computer - White: 0
Human - Black: 3

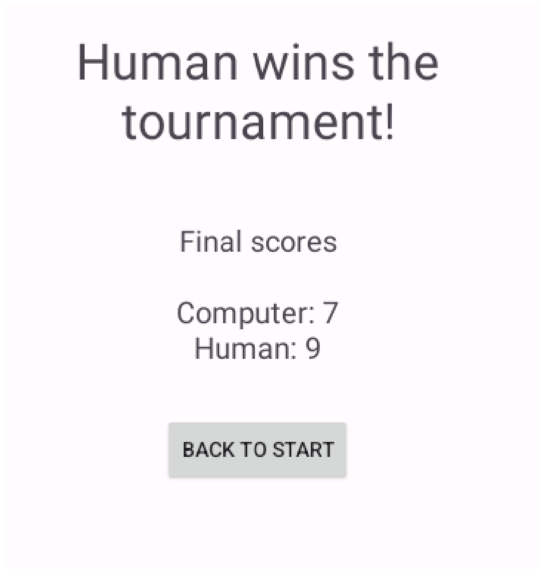
Tournament Score:
Computer - White: 7
Human - Black: 9

Computer - White wins!
Round is over!

The final score details are above!

The score tally results can be found in the game log!

Winner of the tournament being announced

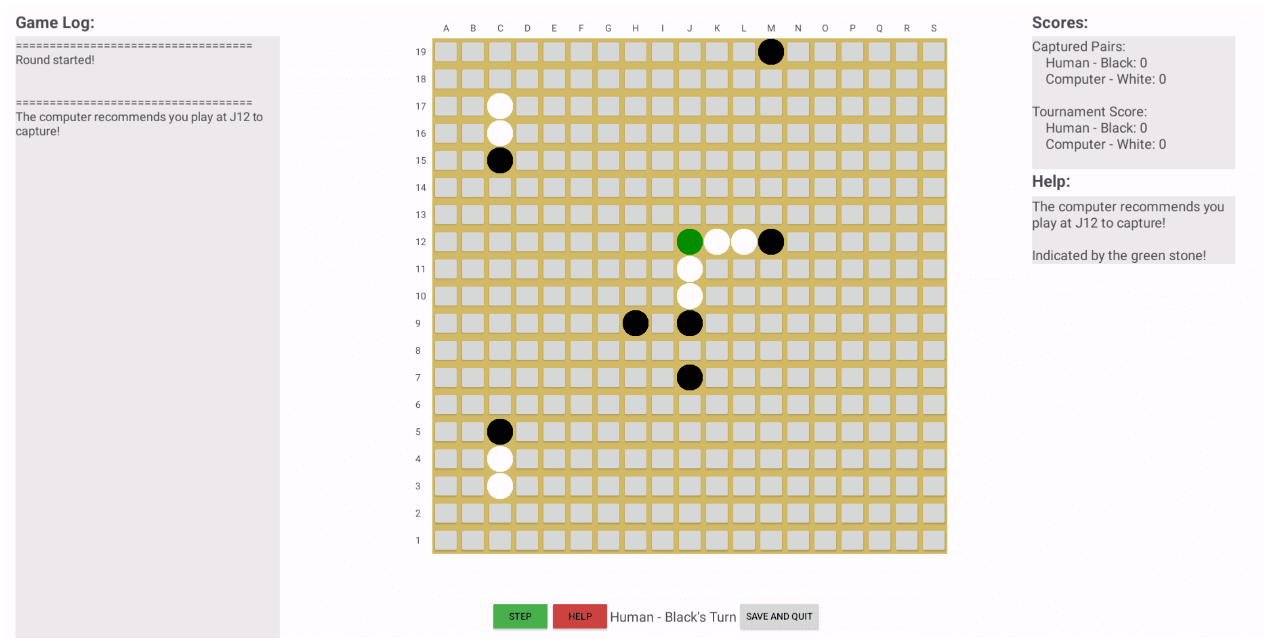


Human wins the tournament!

Final scores

Computer: 7
Human: 9

Representative screenshot of the game:



Game Log:

```
==================================
Round started!

==================================
The computer recommends you play at J12 to
capture!
```

Scores:

Captured Pairs:
Human - Black: 0
Computer - White: 0

Tournament Score:
Human - Black: 0
Computer - White: 0

Help:

The computer recommends you play at J12 to capture!

Indicated by the green stone!