THE MANCALA CORE ANALYSIS AND LEARNING SYSTEM EXPLAINED

By Kalpana Baheti

NOTE:

For experienced readers: please note that all algorithms marked with a '✠' in the Detailed Solution (Page 2) are partially or completely original but are proven and may be reused elsewhere accordingly.

ABSTRACT

The solution presented is an intelligent and personalized game teaching and feedback system which functions through two stages; analysis and conversion to feedback. A brief explanation of the context that has been taken as an example is in order. The game taken as an example to explain the working of the solution is called Mancala. It is a game with six cups accompanied by a home bowl on each player's side. Each of the twelve cups have four marbles in them to begin with. The player must make a choice by choosing one of his/her six cups. Distributing the marbles across all twelve cups and his/her home bowl according to the rules of game, the player that has maximum marbles in his/her home bowl is the winner. The teaching module provides active learning and intentional game variation to improve the abilities of the player in the skill they lack. Identification of this skill (or even a specific tendency) is the responsibility of the analysis stage. The feedback stage is responsible for conveying the knowledge in a manner that will help the player grasp the concept best. Certain points to note regarding the detailed solution that follows next:

- The prerequisites for the functioning of the two stages are stated, after which each of the two stages are explained in detail. The stages are each subdivided into beginners, intermediate and advanced level. Each level is initiated by the general explanation of the technique, followed by the algorithms used (in the form of pseudocode) with a detailed explanation of the reason and consequence of the algorithm steps that are stated prior to it.
- The symbol '<>' used in the conversion to feedback stage indicates a condition in the feedback generation format, or a reference to stored game information in the form of English language, or that a minor process (such as randomization) is run at that point.
- The symbol '*' indicates that the algorithm/methodology/template is customized to the needs of the game but uses a combination of already known algorithms. And the symbol '✠' indicated that the algorithm/methodology/template is entirely original.

Each game state that is saved for future reference is stored in two forms; a list of integers which is stored to the gaming system files and a verbal game-to-game description that is stored to the player's files. The game state description is run by a standard customized existing algorithm that formats the description in a manner that makes it grammatically safe to plug in sentences.

EXISTING WORK

The methodology in question is meant to be applicable to all games involving strategic thinking (irrespective of the complexity of the game) and should also, ideally, be an asset to the field of educational gaming. Hence, a brief on the current standing of strategic and educational gaming systems in terms of technical and pedagogical capabilities, and user-satisfaction is warranted.

We first look at the relevant aspects of mentioned categories of games that only use unstructured artificial intelligence to perform the best opponent moves. Examples of such games would be Dota2 in the field of strategic gaming, and Chess in the field of educational gaming. Such game playing systems resort to machine learning and deep learning algorithms (neural networks in specific) for taking a decision on the move to be simulated. As effective and accurate as they are, there is a lack of actual logical reasoning possible behind the moves that are governed by unstructured AI, making it impossible to convey this learning information to the user. Furthermore, there is always the issue of long training time, overfitting and other such common problems associated with deep learning algorithms. The user-satisfaction with the interface and game-playing capabilities are well above average for most of these games today, but the user-improvement rates are still slow and are usually considered to be of negligible importance making the game's pedagogical capabilities very low.

The second type of strategic and educational games are the ones that can use structured AI to govern best moves to be made as well as if they would have used unstructured AI. An example of such a game is AlphaGo by Google DeepMind. It learns the physics of the game through if-then decision rules and can pinpoint actual reasons behind the moves it makes. There are fewer games of such sort today. They are usually not as visually satisfying but are more effective in directly developing thinking ability in children and with the incorporation of the right teaching methodology, they may provide substantial aid in educating children.

In both categories, though hint generation has been implemented, there is not much progress in personalized game-teaching using natural language reasoning. This sets back the pedagogical and skill-development aspect of such games. The methodology stated thus, is an attempted solution to this issue.

DETAILED SOLUTION

PREREQUISITE INFORMATION:
To explain the working of the teaching module certain specifics about the storage of the players' decisions and game states need to be understood first. The game storage section consists of two parts; one for the players' data and one for system data that stores game states and decisions taken by the simulator. Let us look at the data stored in the players' storage section. Consider a generalized decision series taken by player p_i for game g_j:

$p_i g_j$:

Actual move	: n_1	n_2	n_3	.	.	.	.	n_k
Predicted move	: m_1	m_2	m_3	.	.	.	.	m_k
Flag	: x_1	x_2	x_3	.	.	.	.	x_k

The data above is stored in a matrix. It is generated using the following algorithm:

1. When a player makes a move, that is, selects a cup of marbles in it, the cup number is noted as the player's move and is stored in the row of actual moves. There are six cups on a player's side, hence at every turn the player can choose a cup from 1 to 6.
2. After the actual move of a player is entered, the matrix accepts the value for the predicted move as the cup number that is returned by the prediction module. The prediction module calculates and stores the move that it supposes is the best move the player could have made and the profit associated with it. The calculation of this move and profit is initiated as soon the system has finished its chance and the game state has settled.
3. The third row is value is entered after the player has finished his/her move and the profit of that move is assigned to the corresponding variable. The flag for the corresponding column is then derived.
    a. If $n_a = m_a$, then the flag $x_a = 0$.
    b. Else if predicted profit > actual profit, then the flag $x_a = 1$.
    c. Else, $x_a = -1$.

Explanation:
The significance of the flag value is that if the value is 0, the player's move matched the predicted move and there is no learning required for the player as well as the system. If the value is 1, it means that the profit associated with the predicted move is better than the profit associated with the player's move. This directly implies that the predicted move was the better of the two, since the goal of the game is to accumulate maximum marbles. In this case, the apparent mistake is saved for consideration as feedback to player. The process of saving is done by categorizing the error into a type, updating the count of that type for the player in question, and saving the game state information in the buffer to the system data section. The buffer at that moment contains information of the game state before the player made his/her move. If the value of the flag is -1, this indicates that the player performed a more profitable move than what was predicted by the system. Once again, the previous game state is saved to storage section from the buffer, but this time it is categorized as learning for the system and is considered as appreciation for player. These are the prerequisites for the teaching module to function.
The teaching module is activated at the end of the game, except when it is in move-based mode where the feedback is given right after the player's move is performed or as a hint. Move-based mode is an option only given to novices in the game. But in general, the entire teaching is performed after the game is over.

STAGE 1 (ANALYSIS AND CATEGORIZATION):

The first stage of processing feedback is to analyze the game state corresponding to an erroneous move from the player's side and categorize it into a type of error. The analysis is slightly different for each level of the game:

Beginner's level: The player is learning to calculate the consequences of each move and critically decide which move brings him maximum profit. The analysis also, sticks to a single-round (or a home bonus) move analysis maintaining a relation with the player's capabilities. Hence, in the analysis, the following algorithm is followed:

1. Initiate look-ahead_ error count to 0 when a player registers for the first time and create two empty lists; look-ahead_series and decay-window_series.
2. When the flag = 1:
    a. Refer to respective game state
    b. Note the move and the associated profit in the feedback buffer.
    c. Note the difference between actual profit and predicted profit.
    d. Increment look-ahead_error count and normalize the value.
    e. Save this value of look-ahead_error after the game in the look-ahead_series.
        i. If pattern established, then perform decaying window technique
        ii. Else, continue creating repetitions of game states to test player
    f. If decaying window output is saturating below threshold, that is, consecutive outputs are close to the same value for 1/5 of the total outputs generated, then shift player to Intermediate level.
    g. Save deductions in temporary memory for teaching.
3. When the flag = -1:
    a. If pattern has been established:
        i. Appreciation of move
        ii. Analyze the move, store in strategy section, add to teaching content.
    b. Else:
        i. Analyze the move, store in strategy.

Explanation:
The look-ahead error is the first type of error a player can commit. There are three types; look-ahead error, strategy error and risk analysis. The latter two will be discussed subsequently. In this module, for beginner's level, a player has just started adapting to the game. And from the game's natural course of teaching, the first technique automatically inculcated by the player is the ability to calculate each move possible and judge which will bring in the most profit. For a beginner, he or she will not be accustomed to combining two or more turns of theirs to maximize profit instead of merely analyzing the chances in their present situation. In other words, there arise situations in the game (an example of which will be given duly), where the method to maximize profit does not involve taking maximum benefit that present situation seems to offer, rather it involves making an apparently unprofitable move and then compensating the loss in excess by setting a favorable condition for the next move to be made. A beginner, by definition, would not be able foresee so. If the pattern of his play though, indicates that he/she has mastered this skill, the player will be upgraded to Intermediate level shortly with due appreciation.

In the standard case of a beginner, the look-ahead_error count may fluctuate without a definitive pattern since players start by trial-and-error method. Hence, the count after every game is noted and put into a series for pattern analysis. Once a pattern of decreasing error count starts developing, this indicates that the player is learning to look ahead and take decisions. After a threshold, the decaying window function is set as True. The decay-window_series calculation basically assigns a weightage to error counts of each game, highest to most recent and gradually decaying weights going back in time. This not only acts a numerical indication of whether the player is ready to move to the next level, it also offsets the fluctuations in the player's performance. It also accounts for the rare possibility of a player who is already adept at strategizing and has already mastered look-ahead skills. The reason the decay-window function is only activated after a pattern is established is that due to random fluctuations that occur before the pattern, the decay-window numerical output might accidently proclaim that the player is ready for Intermediate level, when the case is quite the opposite.

Pattern Recognition* and Threshold for using decay-window function*:

1. Collect and record error count of 25 players that can be vouched for as beginners, but with randomly varying IQ levels.
2. Check the fitness of each player personally (must be performed by a domain expert).
3. Once player shows signs of using foresight, perform decaying window technique on the error count, with decay ratio (d) = 10^{-6} and starting from present value cut of the window at the value for the first game that the domain expert confirmed as an improvement in the player.
4. Take the average decay window value of all the 25 players. And take the average window size value of all the 25 players. This gives the initial threshold (t) and initial window size to calculate the threshold (w).
5. With every new player, update the window size followed by the threshold.
6. When a player reaches the threshold, the existence of a pattern is established, hence initiate decay-window function for promotion check and instantiate the decay-window_series for the player.

Decaying Window and Threshold for promoting player:
This function is the same one used in the pattern recognition algorithm except that this is aimed at checking if the player has thoroughly mastered look-ahead whereas the other was meant to check if the player has started to use that skill. Consider a player with a present normalized game error count = c for the game he/she just played. His current decay-window value = S.

$S_{updated} \leftarrow [S*(1-d)] + c$

This time, an average of the decay sum is not taken. Instead each of the players are individually judged. This is carried out by checking if the decay sum changes by a factor 1-d for a consecutive set of values. This would indicate that the learning of the player for the level has saturated. If this sum decreases in such a manner for 1/5 of the total count of values in the series (this ratio is selected based on an intrinsic property of the game that was observed and is axiomatic), the player is promoted to Intermediate level. To be clear, steps are as follows:

1. Enter the calculated sum in the decay-window_series.
2. Check for saturation.
    a. If constant factored decrease in sum is detected for a fixed number of continuous games, then promotion flag is set to True and Intermediate level is activated.
    b. Else, repeat 1 and 2 for consequent games.

Now, that the two algorithms have been trimmed to our needs, we can inspect the material that is to be put in for teaching and appreciation. For teaching, the information that is stored is:

1) The move that the player made
2) Profit associated with player's move
3) The ideal move
4) Profit associated with ideal move
5) First turn or Home bonus?
6) Path of ideal move
7) Equivalent suggestions
8) Focus of next game

And for appreciation, the following is stored:

1) The move that the player made
2) Profit associated with that move
3) Was the system wrong? (Add to stored game state tree)
4) Mastery or fluke? (Dependent on decay sum pattern)
5) Look-ahead or strategy? (Dependent on number of turns utilized)

As seen clearly, teaching is based on mistakes in the current game played and does not bring up a pattern of mistakes on part of the player whereas, appreciation is given based on current game decisions as well as pattern of improvement on player's side. The mastery of beginner's level takes around 20-50 games on an average.

Intermediate and Advanced level: In this set of games, the player starts learning to strategize and may simultaneously learn to measure risks and start understanding how the system is programmed to test him/her. This drawback might be taken advantage of, and hence the algorithm for this level must be able to handle such a situation. But this does not mean that the player must be stopped from being able to perform such an observation, in fact it is crucial for the player to start analyzing from the system's point of view and only after due consideration of this fact, the algorithm must start incorporating a certain degree of randomness to its testing function to put off the player's attempts at taking advantage of the program's structured nature. In the case of strategy, where more than one turn is involved, the earlier method of analyzing each move of present state will not always bring maximum profit. Hence, a finite decision tree is generated. The decision tree lists out a series of moves that the system must perform that are independent of the player's series of moves. The decision tree is ended when there is no branch left for the system to analyze which is independent of the player's moves. When the system starts losing games despite these precautions, it is concluded that the player is finally taking advantage of the program's defects, and hence a certain amount of randomization is incorporated in the updates that are introduced in every game for the benefit of the player. This is to ensure that the decisions on part of the system are sometimes counter-intuitive and misleading on purpose. The following algorithms will explain the strategy as well risk analysis mode.

Strategy*:

```
1. Test-case_fulfilled = False and system-error_count = 0 and player-record_count = 0
   and system-error = [] and player-record = []
2. If flag = 1:
    a. Increment player-record_count of the game.
    b. Refer to respective game-state in game buffer and save the game state.
    c. Build decision tree for that game state. Determine the maximum profit path(s).
       Traverse from root and mark as follows:
        i. If node in question is cup number that gives maximum profit using the
           look-ahead algorithm, then mark as True
        ii. Else, mark as False
    d. Check previous test case fulfillment value:
        i. If True, then explain traversal of randomly selected decision tree based
           on the errors made by player.
        ii. Else, pick smallest decision tree.
    e. Assign test_need = True
3. Else if flag = -1:
    a. Increment system-error_count of the game.
    b. Run through nested-outlier analysis.
        i. If analysis is positive, test_need = random (T/F) and call user_feedback
           function.
        ii. Else, pass.
    c. Save game state from the game buffer. Construct decision tree and mark node
       where the move went awry and consequently the profit of predicted and
       actual output did not match.
4. Save system-error_count to system-error list and player-record_count to player-
   record list.
```

Note that the next algorithm, for risk analysis, will be used for testing the player to check if he has learnt strategy. Hence, the situation is conditioned to help the player learn rather than perform the best moves from the system's side. Hence, certain test cases are generated inside the game randomly. These test cases are engineered based on a prediction of the player's history of using strategy. The prediction is performed by analyzing the games where the player made maximum mistakes, that is, the game where the player-record list shows maximum error count. The present game's situations are purposely driven to imitate the conditions of the game the player made mistakes in. Risk analysis is also used in the situation where the nested outlier analysis result is positive. This will be explained in detail further on. Furthermore, please note that the risk analysis function, unlike strategy and look-ahead, is described in the context of the game being played in real-time and not as an algorithm used after the game is over for the teaching module.

Algorithm for generating relevant test cases ✠ in risk analysis:

1. Check if test_need = True
2. If true, then proceed to step 3 else pass.
3. Check game with maximum player-record_count in the list.
4. Refer game states stored corresponding to the game.
5. At every turn, compare present game state with all game states player handled incorrectly in the past.
    a. If decision tree built on present game state traverses any of these past game states, then select that path over maximum profit path. Note player's response.
        i. If predicted move = actual move, remove corresponding game state from memory and decrease respective game player-record_count in list by 1.
    b. Else:
        i. Construct a set full decision tree each starting from one of the choices on system's side and taking player's moves into account.
        ii. Cut off each branch six nodes from the root.
        iii. Calculate the profit for each branch.
        iv. Prune all branches in all trees save the one with maximum profit for the player in each tree.
        v. Prune all branches without a single False node.
        vi. Choose tree with maximum probability. (Explained in detail later)

Explanation:

1) Strategy-
   To begin with, when a player is newly promoted to Intermediate level, test cases are not engineered in the first game. Test cases are situations built in real-time while the game is in session to trigger the player into using the strategy skill. The game (which is randomly configured) may sometimes offer such scenarios and may sometimes not. Please note that not every turn in the game requires strategy. Most require very sound look-ahead skills alone. Hence, in many games, the player may not have to use that skill at all, and will not be tested until in some randomly configured game, such a situation arises where the skill is required and he/she has failed to use it. In such a case, the flag for that specific move is assigned to a value of 1. After the game, the total number of errors made in the game is noted. At this level, the aggregate error made across all games is not taken since each game's performance is looked at individually. There is no measure that indicates that the player is improving at strategy except for the case when the nested-outlier analysis (which will be elaborated shortly) gives a positive result. Once the error count of the game is recorded in the player-

record list, the game state that existed before the unwise move by the player was made is referred to and a decision tree is constructed. The game state is also saved to memory this time to help engineer test cases.

Construction of the decision tree starts with creating a root node with the move number assigned to it which is unique within a game. Then, until all choices of present game state have been added to the tree and the game is not proclaimed as won or lost, the following is performed:

```
1. Present cup = root node.
2. For present cup on system's side:
     a. For a regular turn:
          i. Create a node referencing the previous node or root in case of first
             round of choices.
         ii. Trace the consequence of choosing the cup, note the destination
             cup and game state.
        iii. Check the look-ahead value of the choice.
               1. If it bears maximum profit, then mark node = True
               2. Else, mark node = False
         iv. Present cup = destination cup.
               1. If present cup value = 0 and player has only one choice of
                  cup, execute step 2c.
               2. Else if present cup value = 0, exit and calculate final profit of
                  system as well as player for leaf node, go to step 1.
               3. Else if, present cup = home, execute step 2b.
               4. Else, run step 2a again.
     b. For home bonus:
          i. Execute step 2.
     c. For player's single choice:
          i. Create a node = Player for the cup on the player's side and
             reference the present cup.
         ii. Trace the consequences, note the destination cup and game state
             along with profit made by player.
        iii. Check destination cup:
               1. If player's destination cup = player's home, destination cup
                  = leaf node, calculate profits of both sides for the branch.
               2. Else, execute 2a.
3. Determine maximum profit branches.
4. If more than one branch has same maximum profit, choose branch with minimum
   player's profit.
```

This decision trees are built within the game in the playing module to make the best set of moves possible. For the teaching module, the reverse decision tree is constructed from the player's point of view to maximize his/her profits and minimize the system's profits. All decision trees are discarded after the feedback session from memory except the ones corresponding to flag = 1 or flag = -1.

The trees corresponding to flag = 1, are saved for teaching section, that is public analysis and conversion to English language, whereas the ones related to flag = -1, are saved for private analysis and nested-outlier check.
Construction of test cases is uses the Risk Analysis algorithm. If the test case in one game has been successfully mastered by the player, the next game will not have a test case. And vice versa, except under one condition. If the nested-outlier analysis shows a positive result, it indicates that the player has learnt to take advantage of the test cases generated. He/ she might purposely play in a manner that triggers the game into generating a foreseeable test case in the next game. If the player has a good memory of how he/she has played in the past, this becomes even more of a challenge to the system to ensure that learning continues. Nested-outlier analysis takes care of this. But before it handles the issue, it allows the player to take advantage, since the identification of the outlier needs a certain number of games to be taken advantage of. Not only that, it gives due appreciation for the skill that is used to beat it. When the analysis result is positive, the test case flag is randomized so that test cases are not generated as expected. Note that by this point in the game, the player and system are equals, and the system chooses to give importance to mutually learning, over winning.

Nested-outlier analysis ✠ is used to check if the player has learnt to use the system's method of testing to his/her advantage. An outlier in a game like Mancala may be described as the few rare times the system has earned a flag value of -1. This will be reflected in the system-error counts in the system-error list. These outliers as such are not a problem. But when these outliers start becoming more frequent it can be concluded that the player is exploiting the teaching tactics of the game. Specifically, the games where the test need held a True value are the games taken into consideration. Hence, the following steps are followed:

1. Run all games with test cases implemented through a standard outlier detection procedure.
2. Note the frequency (with respect to games played) of these outlier counts.
3. Run a standard outlier detection procedure on these frequencies and set fixed boundary limit.
4. When first outlier of the secondary detection is discovered, test need = random

Ideally, there should not be another outlier in the second detection after the first. In case of one, there will be need for an intervention by the domain expert and the developer to analyze where the error exists.

Decision trees that are built in Risk Analysis step 5b, by considering player's moves as well. The difference in the construction of this tree and the one constructed in the Strategy algorithm is that every node on the Strategy tree had a probability = 1. On the other hand, the nodes created for the player's move have a probability less than or equal to 1, based on their past moves. There are three types of tendencies player's usually have when put under a time pressure in a game like Mancala:

1) Tendency to gain Home Bonus
2) Tendency to distribute rather than make an unprofitable and inconsequential move
3) Tendency to look several steps ahead in a regular turn

4) Tendency to trick the system based on knowledge of how the system works

All players' statistics are judged based on their performance in their test cases and their respective probabilities are calculated, hence giving each node on player's side, a value between 0 and 1. The leaf node of the branch bears a product of all the probabilities on that branch starting from root. The branch with maximum probability is selected and the first move on that branch is performed from the system's side.

For teaching, the information that is stored is:

1) The branch that the player chose
2) The profit associated with the player's branch
3) The predicted branch
4) Profit associated with predicted branch
5) Strategy or look-ahead error?
6) Path of each node to the next on the ideal branch
7) Equivalent suggestions
8) Alert on test case? (yes/no)

For appreciation, the information that is stored is:

1) The branch that the player chose
2) Profit associated with it
3) Mastery or fluke? (Dependent on player-record list)
4) Strategy or Risk Analysis? (Dependent on outlier analysis)

NOTES ON STAGE 1:

1) Moves involved in test cases are never given a flag value of -1, even if the system does not make maximum profit. They are assigned to 0 if the player performs correctly, else they are assigned to value -1.
2) The failure of generating a test case that predicts the player's moves accurately and the randomization of the test case existence results in creating unpredictability and complexity in the system primarily aimed at putting off the player's attempts to exploit the test case feature.
3) Along with the different levels, there are multiple configurations of the game, which change and randomize the number of marbles in the cup rather than sticking to standard four marbles per cup. This cycle too is based on the indication of the player's improvement.

Analysis and categorization of skills is handled by stage 1. Stage 2 aims at converting the results of stage 1 to clear simple and intelligible explanations in the English language. It also, to some extent, allows interaction with player and graphical aid for player's understanding.

STAGE 2 (CONVERSION TO FEEDBACK LANGUAGE, INTERACTION, AND LEARNING)

In this stage, the stored information from the levels of the previous stage is converted into explanations in English. To ensure that the instruction and teaching is as genuine and lively as possible, popular as well as customized datasets of phrases are used and sentences of several variations are constructed along with the necessary information components. There is a possibility that the player did not understand a move or aspect of the game. In this case, the module maps the keywords in the player's request for information. Then it checks the rule database in case the request is regarding a rule. Else it attempts to pinpoint which move or game state the player is referring to. Either ways, before providing further explanation, the system confirms with the player if the understanding of what information is requested is the same.

Beginner's level: At this level, the amount of feedback given varies over the entire course. Before the pattern of applying the skill is established, the feedback covers only the first case of flag = 1, followed by a detailed explanation of the game state, the ideal move and the profit that could have been gained. After the pattern is established, the feedback mentions all moves in the game where errors occur and brief about them collectively at the end in terms of profit. The number of moves to speak of would not be many considering that the player is already learning to master the look-ahead skill. Once the decay count crosses the threshold, the moves are merely mentioned as better moves that could have been done.

Template:

Part 1 –
Game Status

If game won:

Depending on the margin of the win the first part of the feedback is the compliment. Now it is very important that this part is done as genuinely as possible because as a teacher, it is responsible for making the player (which will most likely be a child) feel like it has achieved something of real significance even though it is mere game.

A database of such compliments does not exist but it is possible to create one or segregate a collection from those directly available on the net. the collection will consist of sentences such as:

"Well, you should be proud of yourself. Not everyone gets the hang of this so quick."
"Ah! That was a narrow lose for me. But I have to admit that those moves were good ones." (For a marginal win)
"You beat me hands down. That's that."
"You're the man! Let's have another go hmm?"

If game lost:

This section is even more important since the player must not feel discouraged by the outcome of the game. It is a good practice to point out something positive the player managed to do in the game he/she lost. The statement collection will be of the following nature:

“Hmm. You did well nevertheless. Especially that move where you chose *<random move that turned out well for the player>* when our game was at the point where *<description of game state associated with that move + visual aid>*. That was good. Okay now let’s see, …”

“Well, your foresight is quite sound and everything seems to be working fine otherwise. This small thing though sets you back in this game. Let’s resolve it so we can have better games”

The ones above are for the first or second time a player loses the game after a pattern is established. Before that stage and after, the feedback is of following nature:

“Good game well played. Happens sometimes. I learnt a lot too from the game. Now before we move to another game, I have to tell you something though.” (for the games before pattern was established)

“Great game mate. Love the move where you picked *<select move with maximum profit>* and it caused that thing where *<description of game state associated with that move + visual aid>*. It was pretty cool.”

Part 2 –
Critical advice and improvement action ✠

Teaching section for players whose pattern has not yet been established:

For such cases, the feedback takes one example from the game (by default it takes the first) and explains the reason it was not the best move to make and what could have been done instead.

“when you picked *<move picked by player>*, what happened was that *<game state description and visual aid>.* And that gave you a profit of *<associated profit>* marbles, if I am not mistaken. Now let’s suppose you picked *<predicted move>*, then what would have happened is *<path description with visual aid>*. That might have just been better, right?”
<If home bonus = True>
“This time you had a home bonus, wow! You would earn more marbles in your home with this move. We looked ahead at every turn we have and do what will earn us the most marbles, right? Next game, try it out! It’s pretty cool.”
<For equivalent suggestions>
“You see, there was another choice that would earn you as much. It was *<Equivalent choice>*. Would you like to see how that would have turned out?”
<If yes>
“*<Description of consequences with visual aid>”*
<If no>
“Okay, you’ve got this then.”
<Focus of next game>

"The next game is going to be a better win, just remember what we discussed okay? You ready?"

Teaching section for players whose pattern has already been established:

For such cases, the errors are listed together and a reminder that the look-ahead skill ought to be used is given. Then it prompts the player giving the option for further clarification for any one of the errors.

"Okay so there were a couple of places that could have gone differently."
<Randomize between sentence form and listed form>
<Sentence format>
One, was where *<player's move 1>* was picked when the game looked like *<game state description>*, maybe a better choice would've been *<predicted move>*. It brings *<ideal profit>* marbles. Then another instance was when… "
<Listed format>
"1. The move where… <> when… <> could have been replaced by… <> and it would bring you the profit… <>.
2….
3…."
<Reminder>
"Well, I know you have got the hang of it. I have watched you play great games. We are going to practice a little more though. You just need to use your foresight more often. Constant Vigilance! Would you like me to explain any of those moves I listed above? Or we are good to go…?"
<If yes ask which move and explain>"

Teaching section for players who have crossed the decay value threshold:

"There not much left to teach you now. You will soon move on to greater challenges. A few notes here and there. *<player's move selected at random>* could have been replaced by *<predicted move>* maybe. Check the game state out if you like; *<game state description>*. Just keep doing your best, and you are going to progress to the next level in a wink of an eye. good luck next game and good game, once again."

Part 3 –
Recognition of strengths ✠

Appreciation for players before their pattern is established:

"That move where you chose *<player's move>* when we were at this point where *<game state description>*. You got me there. You got *<profit>* marbles. Nice one! I should learn it from you. Let's see!"

Appreciation for players whose pattern is established:

"I really want to congratulate you on your move right there; *<player's move>* when we were at this point where *<game state description>*. You got me there. Nice one!"
<Strategy = True>
"That was something beyond my skills. I think you are getting ready to move on to the next stage. You have almost achieved mastery here. Do you know what was special about that move?" *<Implement analysis of strategy module and explain>*

Intermediate and Advanced level: Players of this level are already quite adept at the game. The explanation of strategy is randomly incorporated in some feedback sessions while it is absent in some. This is followed to give the player a chance to learn the skill on his/her own and avoid spoon-feeding. Once the outlier analysis results turn out to be positive, the teaching entirely stops and the interactive sessions become more active. In the rare case where the outliers continue to occur, over a span of games (which has not yet been calculated for this project), the program automatically ceases to execute and the player is presented with a digitally signed certificate as proof of mastery of all skills that the game demands.

Part 1 –
Game status

If game won:

"Good game, well played. Awesome moves this time!"
And other such comments.

If game lost:

"Oh well, we will have another round if you like. You were good though. Just a bit more alertness I believe. That should do the trick."
And other such comments.

Part 2 –
Controlled hinting ✠

Teaching section for players who have not yet learnt Risk Analysis:

"Check this scenario where *<description of game state>*. Now your moves were fine till here! And then there was this move *<ideal move on decision tree>*."
<If node was a false node>
"I know it doesn't seem like the best move since it does not instantly seem to give us many marbles but just suppose we picked it! Now, *<description of consequences along the ideal branch>*. That is what would have happened. Now is that not a better move?"
<If node was true node>
"Well I guess you need to wake up and start playing for real mate! You can play much better than this c'mon now! Why would you make this move *<player's move>* when your game was this good for you; *<game state description>*!"

Part 3 –

Recognition of mastery ✠

Appreciation section for Intermediate and Advanced level:

“You are mastering this game. I did not see this move coming. *<Player’s move>* in this tight corner *<game state description>* was awesome. If you don’t mind, could you run me through your thought process? Maybe you could tell me why you picked that move?”

<Hand over control to interaction module>

Part 4 —
Test Alert

If test need = True and Alert = True:

“You are almost there okay! You are going to be tested on something similar, so watch out in your next game.” And other such comments.

Interaction module ✠: The interaction module is common for all levels and is a subset of the feedback module. It prompts the player into asking questions, it confirms the request from player, and especially for the Risk Analysis stage, it requests the player to divulge his/her thought process. To acquire information from the answers that can then be incorporated in the analysis and categorization procedure, the following rules are followed for all answers:

1. All forms of “yes” are noted as True. The reply means that further explanation is required or that the next game is ready to begin. There are a few exceptional cases covered where a question about the risks the player takes is asked. It is close-ended question. In such cases, if the answer is “yes”, results in the assignment of test need to random.
2. All forms of “no” are noted as False. It means that no further explanation is required. Or it results in the game being switched off with all the states saved.
3. All moves in the games may be referenced along with their game states in the player’s reply. The move is then analyzed. The set of words in explanation form are mapped to a path of moves. This initially will need intervention from domain experts to help facilitate understanding, but over time, the system will learn to handle more diverse path explanation statements.
4. Sets of words in the player’s explanation will be identified to trigger the test need value to random. For example, any explanation from player containing a set of words like “You usually do <>“ or “I knew <> was going to happen/occur/take place…” are examples of sets of words that trigger the test need to a random Boolean value. This again, gets better with experience given by domain experts and more players.
5. After two attempts of confirming what the player means and if both replies were negative, irrespective of what the system learnt from it, a standard answer of “Okay, cool!” or “Hmm. Thanks.” or something similar will be given.

NOTES ON STAGE 2:

1) These are merely examples of the statement framework. Ideally, it must randomize its selection of these phrases and must have an entire back-end module that is created for learning newer and genuine compliment-phrases for won games from the real world, alike to systems like Alexa and Siri.
2) The framework may be defined in the form of a Context Free Grammar individually structured for each case.
3) The Interaction module behaves and follows the same methods of NLP as do Alexa and Google Assistant except that the understanding is focused on capturing game-related terminology and on converting a set of words to an indicator of a specific game path.

This is complete solution of the functioning of the teaching module of the game.

PROBLEMS SOLVED

The following observations about recent advances in gaming and pedagogy must be noted before examining the novelty of presented solution:

1. There are several genres of games that are played today. Some are biased towards problem-solving and strategy and some are biased towards entertainment and player experience. The more complex the design of such games become, the greater is the challenge to simulate the best moves possible. In the light of recent advances, the simulation of smart playing heavily relies on neural networks and other heuristic-based algorithms of machine learning. The absence of proof regarding why certain decisions are profitable and why some are not, is inherent in stated techniques. Hence, although the system may be able to play intelligently, it is unaware of the 'reason' behind its success (or failure). This makes it impossible for the system to be able to give personalized and effective teaching and training to the player. The solution documented is a well-reasoned example of a method to engineer automated teaching modules for every game that has a certain logic required in its playing.
2. Mancala is an enjoyable and understated game that is overshadowed by more popular games such as chess. Nevertheless, it demands of the player to extensively use his/her look-ahead, strategy and risk analysis skills, almost as much as chess does and at the same time, it's rules are simple to understand. The implementation of personalized game-training modules has failed in more complex games, due to two reasons; one, is that game developers focus more on designing better games rather than on creating better players and two, the training module is a project under natural language reasoning, a field that has not yet made significant progress. The design of such a module for the Mancala game has not been considered for similar reasons. This solution offers precisely that insight.
3. Teaching and training in general, has been compromised upon substantially. And learners are deprived on several occasions of the enthusiasm to learn because of the mechanical nature of today's learning systems. Games are made to beat the player, not teach the player how to think better, or engage the player in a fruitful game discussion. In teaching strategy, it is important that the opponent is purposely allowed

to win at times, because it gives information on the opponent's tendencies and flaws. The same technique can at times help boost the confidence of the player and keep him/her engaged. Games inherently are not trained to take a fall, with the intention of achieving a long-term win; such as producing more astute players. That is still a human virtue alone. The documented model incorporates this behavior and several others that train the player to be keener in an all-round manner and help the player unconsciously develop skills that are in high demand in the real world.

INVENTIVE STEPS

1. It is evident that several familiar algorithms have been customized and combined to a good measure along with some additional steps unique to the game, to complete the solution for the game teaching module. **However, put together in the manner described, it embodies an instructive, interactive, user-friendly and personalized teaching and training module for game players.** The algorithms that have been customized and redesigned for teaching purposes are:
    a. Skill analysis methodology: beginner's level – Uses decay window function with statistically derived thresholds to assess changes that must be implemented in game teaching with the aim of improving the player's skills.
    b. Test case construction methodology: advanced level – Uses a systematic set of rules to generate from scratch or from a conveniently close game state, a past scenario that was not properly handled by the player.
    c. Nested outlier detection algorithm: advanced level – Used to detect outliers in the frequency of the outliers of a data stream.
    d. Template for instruction module: Beginner's, intermediate and advanced level – Standards for instructing and engaging with the player in an effective and highly personalized and human manner.

    The game playing module (which is already invented and is not elaborated upon here) need not make use of structured learning, but the algorithms of the game training module make use of structured learning alone, except the test case determination and a specific step in the risk analysis method. These exceptions are also utilized to provide incentive for the player to beat the game which will take remarkable cognitive and strategy skills, the development of which is the fundamental goal of the game. Furthermore, the game uses nested outlier analysis, a completely customized technique, that considers game losses on the system's side as a good sign and an opportunity to test the player better, as the algorithms clearly state. **These techniques set the solution apart from all such educational gaming solutions. Also, the teaching module is an entirely original introduction to online strategy games.**

2. The instruction module streamlines comments from a database of compliments for players who have won or lost and a database of general comments regarding game progress and encouragement. It focuses on sounding versatile, positive and natural as well as conveying the necessary improvements that the player could make. It controls the amount of advice and explanation to impart and the praise/criticism to give based

on the history and status of the player and the nature of the game just played. **The novelty is in the adherence to this discipline while designing automated game teaching modules.**

3. The interactive session is trained to find game related words and improves with experience. Here, unstructured learning algorithms are used to understand the players' needs better. Questions and confirmations are pursued to pinpoint the subject of the discussion. **The responses refrain from giving the impression that the player is difficult to comprehend or that the algorithms have failed to capture required information. It mandatorily gives assent eventually in a satisfactory manner, very much like humans do. This makes the system more user-friendly.**

4. While it is true that some games tend to automatically become more complex with time, the skill set used for the games is still unlikely to change. **The algorithms are designed as per the skill needed which means that this work is potentially a foundation for creating entire teaching modules for much more complex games that progress and evolve on their own with time.**