AI Model for Computer games based on Case Based Reasoning and AI Planning

Vlado Menkovski
Athens Information Technology
0.8km Markopoulou Ave.
Peania, 19002, Greece
vmen@ait.edu.gr

Dimitrios Metafas
Athens Information Technology
0.8km Markopoulou Ave.
Peania, 19002, Greece
dmeta@ait.edu.gr

Abstract

Making efficient AI models for games with imperfect information can be a particular challenge. Considering the large number of possible moves and the incorporated uncertainties building game trees for these games becomes very difficult due to the exponential growth of the number of nodes at each level. This effort is focused on presenting a method of combined Case Based Reasoning (CBR) with AI Planning which drastically reduces the size of game trees. Instead of looking at all possible combinations we can focus only on the moves that lead us to specific strategies in effect discarding meaningless moves. These strategies are selected by finding similarities to cases in the CBR database. The strategies are formed by a set of desired goals. The AI planning is responsible for creating a plan to reach these goals. The plan is basically a set of moves that brings the player to this goal. By following these steps and not regarding the vast number of other possible moves the model develops Game Trees which grows slower so they can be built with more feature moves restricted by the same amount of memory.

Categories and Subject Descriptors

I.2.1 [Applications and Expert Systems]: *Games*

General Terms

Algorithms, Performance.

Keywords

Game AI, Case Based Reasoning, AI Planning, Game Trees

1. Introduction

The goal of this effort is to explore a model for design and implementation of an AI agent for turn based games. This model provides for building more capable computer opponents that rely on strategies that closely resemble human approach in solving problems opposed to classical computational centric heuristics in game AI. In this manner the computational resources can be focused on more sensible strategies for the game play.

With the advancement in computer hardware increasingly more computing power is left for executing AI algorithms in games. In the past AI in games was mainly a cheating set of instructions that simulated the increasing difficulty in the game environment so that the player had the illusion of real counterpart. Improvement in available memory and processing power allows implementation of more intelligent algorithms for building the game environment as well as direct interaction with the human players.

In this particular research the emphasis is put on the interaction between the AI agent and a computer player in the realm of the game rules. It is particularly focused on turn based games that have the elements of uncertainty like dice or concealed information. At the beginning a description of Game AI algorithms are given; such as Game Trees and Minimax. The following section describes an approach of using AI Planning to improve building Game Trees in games with imperfect information where Game Trees tend to be very large with high growth ratio. Section 4 discusses another approach that provides a significant reduction to the number of considered moves in order to find the favorable strategy of the AI player. This approach uses AI Planning techniques and Case Base Reasoning (CBR) to plan for different scenarios in predetermined strategies which would be analogous to human player experience in the particular game. The CBR database illustrates a set of past experiences for the AI problem and the AI Planning illustrates the procedure to deal with the given situation in the game. In the next two sections implementations and evaluations of both approaches are given. The AI Planning approach is implemented with the Tic-tac-toe game and the combined AI Planning and CBR approach is implemented with a model for the Monopoly game. The last part contains conclusions and future work ideas.

2. Game Trees and Minimax

Game Trees are common model for evaluating how different combinations of moves from the player and his opponents will affect the future position of the player and eventually the end result of the game. An algorithm that decides on the next move by evaluating the results from the built Game Tree is minimax [1]. Minimax assumes that the player at hand will always choose the best possible move for him, in other words the player will try to select the move that maximizes the result of the evaluation function over the game state. So basically the player at hand needs to choose the best move overall while taking into account that the next player(s) will try to do the same thing. Minimax tries to maximize the minimum gain. Minimax can be applied to multiple levels of nodes on the game tree, where the leaves bring the final known (or considered) game state.

The minimax theorem states:

For every two-person, zero-sum game there is a mixed strategy for each player, such that the expected payoff for both is the same value V when the players use these strategies. Furthermore, V is the ***best*** *payoff each can expect to receive from a play of the game; that is, these mixed strategies are the optimal strategies for the two players.*

This theorem was established by John von Neumann, who is quoted as saying "As far as I can see, there could be no theory of games … without that theorem … I thought there was nothing worth publishing until the *Minimax Theorem* was proved" [2].

A simple example of minimax can be observed by building a game tree of the tic-tac-toe game. The tic-tac-toe game is a simple game which can end by the first player wining, the second player wining or a tie. There are nine positions for each of the players in which at each turn the player puts X or O sign. If the player has three adjacent signs in a row, column or the two diagonals he or she wins. This game has limited number of position and it is well suited for building the whole game tree. The leaves of this tree will be final positions in the game. A heuristics evaluation function will also need to be written to evaluate the value of each node along the way.

3. AI Planning for building Game Trees

3.1.1 AI Planning

AI Planning also referred as Automated Planning and Scheduling is a branch of Artificial Intelligence that focuses on finding strategies or sequences of actions that reach a predefined goal [3]. Typical execution of AI Planning algorithms is by intelligent agents, autonomous robots and unmanned vehicles. Opposed to classical control or classification AI Planning results with complex solutions that are derived from multidimensional space.

AI Planning algorithms are also common in the video game development. They solve broad range of problems from path finding to action planning. A typical planner takes three inputs: a description of the initial state of the world, a description of the desired goal, and a set of possible actions. Some efforts for incorporating planning techniques for building game trees have also shown up, similar to the approach explored in this effort. In addition Cased Based Reasoning [4] techniques are also gathering popularity in developing strategies based in prior knowledge about the problems in the games. One of the benefits from Hierarchical Task Network (HTN) [5] planning is the possibility to build Game Trees based on HTN plans; this method is described in the following section.

3.2 Game Trees with AI Planning

An adaptation of the HTN planning can be used to build much smaller and more efficient game trees. This idea has already been implemented in the Bridge Baron a computer program for the game of Contact Bridge [6].

Computer programs based on Game Tree search techniques are now as good as or better than humans in many games like Chess [7] and checkers [8], but there are some difficulties in building a game tree for games that have imperfect information and added uncertainty like card or games with dice. The main problem is the enormous number of possibilities that the player can choose from in making his move. In addition some of the moves are accompanied with probabilities based on the random elements in the games. The number of possible moves exponentially grows with each move so the depth of the search has to be very limited to accommodate for the memory limitations.

The basic idea behind using HTN for building game trees is that the HTN provides the means of expressing high level goals and describing strategies how to reach those goals. These goals may be decomposed in goals at lower level called sub-goals. This approach closely resembles the way a human player usually addresses a complex problem. It is also good for domains where classical search for solution is not feasible due to the vastness of the problem domain or uncertainties.

3.2.1 Hierarchical Task Networks

The Hierarchical Task Network, or HTN, is an approach to automated planning in which the dependency among actions can be given in the form of networks [9] [Figure 1].

A simple task network (or just a task network for short) is an acyclic digraph $w = (U, E)$ in which U is the node set, E is the edge set, and each node $u \in U$ contains a task t_u. The edges of w define a partial ordering of U. If the partial ordering is total, then we say that w is totally ordered, in which case w can be written as a sequence of tasks $w = \langle t_1, t_2, \ldots, t_k \rangle$.

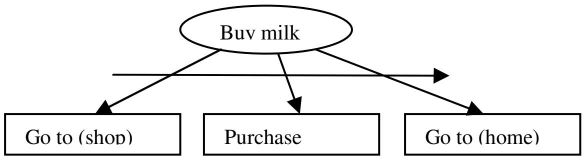

Figure 1: Simple Hierarchical Task Network

A Simple Task Network (STN) method is a 4-tuple of its name, task, precondition and a task network. The name of the method lets us refer unambiguously to substitution instances of the method, without having to write the preconditions and effects explicitly. The task tells what kind of task can be applied if the preconditions are met. The preconditions specify the conditions that the current state needs to satisfy in order for the method to be applied. And the network defines the specific subtasks to accomplish in order to accomplish the task.

A method is relevant for a task if the current state satisfies the preconditions of a method that implements that task. This task can be then substituted with the instance of the method. The substitution is basically giving the method network as a solution for the task.

If there is a task "Go home" and the distance to home is 3km [Figure 2] and there exists a method walk-to and this method has a precondition that the distance is less than 5km, then a substation to the task "Go home" can be made with this method instance.

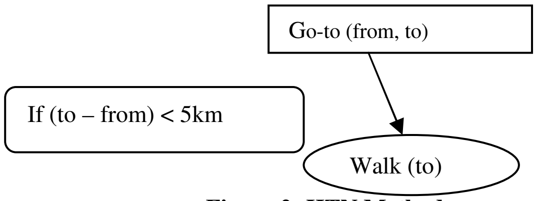

Figure 2: HTN Method

If the distance is larger than 5km another method instance needs to be substituted [Figure 3].

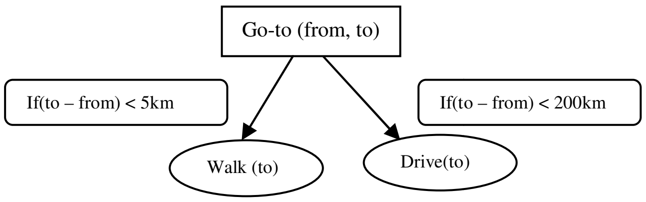

Figure 3: HTN Method 2

An STN planning domain is a set of operations O and a set of methods M. A STN planning problem is a 4-tuple of the initial state S_0, the task network w called initial task network and the STN domain. A plan $\pi = \langle a_1, \ldots, a_n \rangle$ is a solution for a planning problem if there is a way to decompose w into π if π is executable and each decomposition is applicable in the appropriate state of the world. The algorithm that is capable to decompose these networks into plans is called Total-forward-decomposition (TFD) [9] or Partial-forward-decomposition (PFD). However there are cases where one does not want to use a forward-decomposition procedure. HTN planning is generalization of STN planning that gives the planning procedure more freedom about how to construct the task networks.

In order to provide this freedom, a bookkeeping mechanism is needed to represent constraints that the planning algorithm has not yet enforced. The bookkeeping is done by representing the unenforced constraints explicitly in the task network.

The HTN generalizes the definition of a task network in STN. A task network is the pair $w = (U, C)$ where U is a set of task nodes and C is a set of constraints. Each constraint in C specifies a requirement that must be satisfied by every plan that is a solution to a planning problem.

The definition of a method in HTN also generalizes the definition used in STN planning. A HTN plan is a 4-tuple of name, task, subtasks, and constraints. The subtasks and the constraints form the task network. The HTN planning domains are identical to STN planning domains except they use HTN methods instead of STN methods.

Compared to classical planners the primary advantage of HTN planners is their sophisticated knowledge representation and reasoning capabilities. They can represent and solve a variety of non-classical planning problems; with a good set of HTNs to guide them, they can solve classical planning problems orders of magnitude more quickly than classical or neoclassical planners. The primary disadvantage of HTN is the need of the domain author to write not only a set of planning operators but also a set of methods.

3.2.2 HTN Planning in building Game Trees

For a HTN planning algorithm to be adapted to build game trees we need to define the domain (set of HTN methods and operators) which is the domain of the game. This is in some sense a knowledge representation of the rules of the game, the game environments and possible strategies of game play.

In this domain the game rules as well as known strategies to tackle specific task are defined. The implementation of Game Tree building with HTN is called Tignum2 [9]. This implementation uses a procedure similar to forward-decomposition, but adapted to build up a game tree rather than a plan. The branches of the game tree represent moves generated by the methods. Tignum2 applies all methods applicable to a given state of the world to produce new states of the world and continues recursively until there are no applicable methods that have not already been applied to the appropriate state of the world.

In the task network generated by Tignum2, the order in which the actions will occur is determined by the total-ordering constraints. By listing the actions in the order they will occur, the task network can be "serialized" into a game tree [Figure 4] [Figure 5].

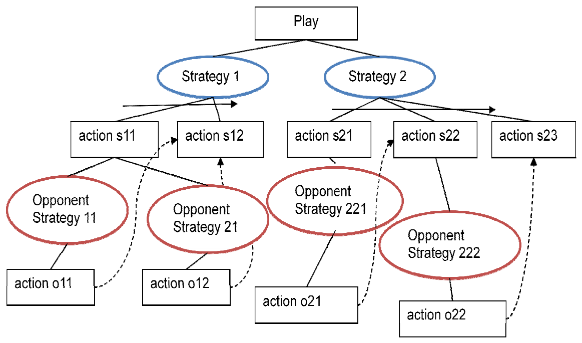

Figure 4: HTN to Game Tree Algorithm

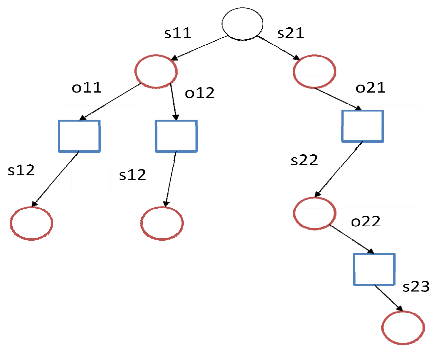

Figure 5: Game Tree built from HTN

4. Case Based Reasoning in Game Strategies

4.1 Case Based Reasoning

Case-based reasoning (CBR) is a well established subfield of Artificial Intelligence (AI), both as a mean for addressing AI problems and as a basis for standalone AI technology.

Case-based reasoning is a paradigm for combining problem-solving and learning that has became one of the most successful applied subfield of AI of recent years. CBR is based on the intuition that problems tend to recur. It means that new problems are often similar to previously encountered problems and, therefore, that past solutions may be of use in the current situation [10].

CBR is particularly applicable to problems where earlier cases are available, even when the domain is not understood well enough for a deep domain model. Helpdesks, diagnosis or classification systems have been the most successful areas of application, e.g., to determine a fault or diagnostic an illness from observed attributes, or to determine whether or not a certain treatment or repair is necessary given a set of past solved cases [11].

Central tasks that all CBR methods have to deal with are [12]: "to identify the current problem situation, find a past case similar to the new one, use that case to suggest a solution to the current problem, evaluate the proposed solution, and update the system by learning from this experience. How this is done, what part of the process that is focused, what type of problems that drives the methods, etc. varies considerably, however".

While the underlying ideas of CBR can be applied consistently across application domains, the specific implementation of the CBR methods –in particular retrieval and similarity functions– is highly customized to the application at hand.

4.2 CBR and Games

Many different implementations of CBR exist in games. CBR technology is nicely suited for recognizing complex situations much easier and more elegant than traditional parameter comparison or function evaluation. There are especially evident cases in real time strategies where different attack and defense of global strategies are nicely defined by CBR datasets and later used in the running games. Also intelligent bots behavior is also another typical example. Depending on the number of enemy bots the layout of the terrain and position of human players the CBR system finds the closest CBR case and employs that strategy against the human players which in prior evaluation was proved to be highly efficient.

5. Game Trees with AI Planning – Tic-tac-toe

In order to show the expressive power of AI Planning in defining strategies for games, and the use of these plans to build Game Trees I implemented an algorithm that builds Game Trees for the Tic-Tac-Toe game.

The game tree of Tic-Tac-Toe shows 255,168 possible games of which 131,184 are won by X (the first player), 77904 are won by O and the rest 46,080 are draw [13]. All these games can be derived from building a complete Game Tree.

Even though it is possible to build a complete game tree of Tic-tac-toe it is definitely not an optimal solution. Many of the moves in this tree would be symmetrical and also there are a many moves that would be illogical or at least a bad strategy to even consider.

So what strategy should X (the first player) choose in order to win the game?

There are few positions that lead to certain victory. These positions involve simultaneous attack on two positions so the other player could not defend, basically the only trick in Tic-Tac-Toe.

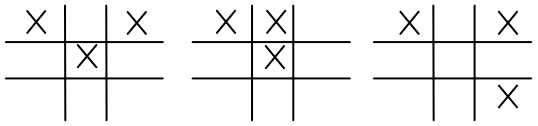

Figure 6: Tic-tac-toe winning strategy positions

Position 1 leads to victory if the two of the three fields: top middle, bottom left corner and bottom right corner are free [Figure 6].

Position 2 lead to victory if two of the three fields: top right corner, bottom right corner and bottom middle are free [Figure].

And in the third position if the two of center, middle top and middle left are available the position is a certain victory.

There are many different arrangements of the player's tokens that give equivalent positions as these three positions. By using planning we do not need to consider all possible layouts but just consider these three similar to what a human would consider.

The game starts from an empty table.

The two relevant strategies that would lead to these positions are to take one corner or to take the center [Figure 7].

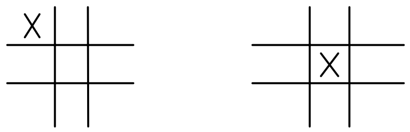

Figure 7: Tic-tac-toe Two starting moves

The center position as we can see in the simulation results lead to a bigger number of victorious endings but it is also a straight forward strategy with obvious defense strategy.

At this point we need to consider the moves of the opponent. If we take the left branch the opponent moves can be a center, a corner or a middle field. We also need to differentiate with a move to a corner adjacent with our like top left or bottom right or across the center to bottom right [Figure 8].

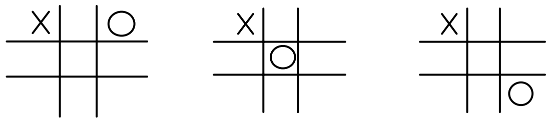

Figure 8: Tic-tac-toe opponent response to corner move

In cases one and two, we have a clear path to executing strategy 3 so we need to capture the diagonally opposite field. And as for the third case the best way to go is to capture the center and go for strategy 1 or 2 depending of the opponent's next move.

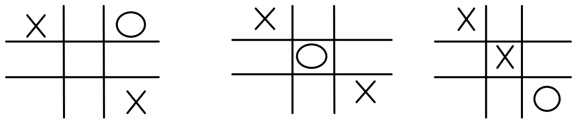

Figure 9: Tic-tac-toe move 2 after corner opening

The first move leads to certain victory, O will have to go to the center and X will achieve strategy 3 [Figure 9]. The second move is a possible way to strategy 3 if O makes a mistake in the next loop, so X goes to the opposite corner. For the third case since O is playing a valid strategy the only move that leaves a possible mistake from O would be to take the center and wait for O to go to the middle and then achieve strategy 1 or 3 which will be a symmetric situation to the one that we will find if we branched with the center.

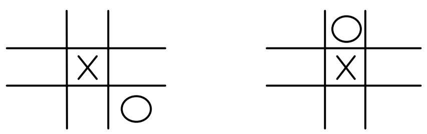

Figure 10: Tic-tac-toe opponent response to center move

If we go back to the second branch [Figure 10], a possible way for the second player to engage is corner or middle. The first move is a valid strategy for O and can be meet with a opposite corner move from X to try a mistake from O in the future exactly the same as in the third case above from the previous branch, and another move would be go to the middle where X eventually achieves strategy 1 or 2.

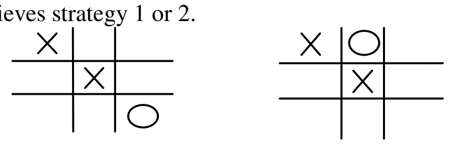

Figure 11: Tic-tac-toe Move 2 after center opening

The fist move will lead to win if O moves to the middle or a draw if it goes for the corners [Figure 11]. In the second case O has to block the lower left corner which leaves X to go for the middle left or corner left which are strategy 1 and 2.

To sum the strategies for the planning, first we have center or corner strategy for the beginning. Then for the center we try to get the corners with the particularly the one opposite to the one O holds. If the center is empty for the second strategy we go for it or we go for the opposite corner. After this point we either block the opponent or try to implement strategies 1, 2 or 3 which lead to victory.

Plan 1: Take center
Preconditions: Center empty
Plan 2: Take corner
Preconditions: All corners empty
Plan 3: Take corner after center
Preconditions: We have center take corner opposite to the one the opponent has
Plan 4: Take diagonal corner
Preconditions: We have a corner, the opponent has the ce-nter and the corner opposite to the one we have is free.
Plan 5: Block
Precondition: The opponent has tree tokens in a row, colu-mn or di agonal
Plan 6: Win
Preconditions: We have two tokens in a row, column or dia-gonal a nd the third place is free
Plan 7: Tie
Preconditions: If all places are taken, it's a tie.

5.1 Hierarchical Task Network

Top level task is Play [Figure 12]. This is a complex task and can be derived into: Win, Block, Tie or Search for Plan. The Search for plan is derived to both Plan 1 and Plan 2 or Plan 3 and Plan 4, which later leads to a call for the opponent's move and a recursive call to Play.

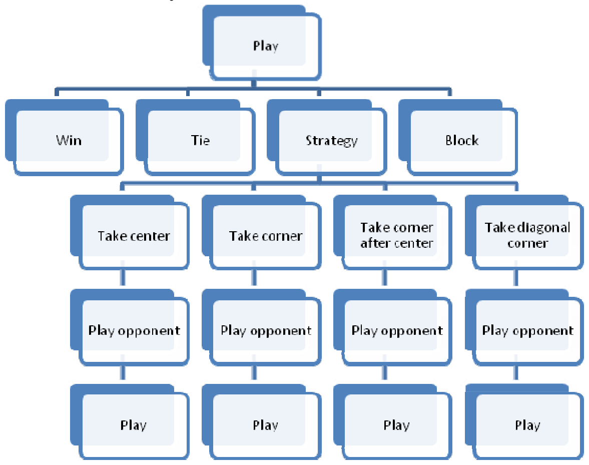

Figure 12: Tic-tac-toe HTN

This HTN when executed will result with plans for possible game scenarios. By creating nodes from each position and linking them with branches with the move of the player we create a game tree for the Tic-tac-toe game over which we can run the minimax algorithm.

This set up with 7 plans with 3 target strategies creates a tree for Tic-tac-toe which considers all possible moves for the second player with only 457 games, 281 of which X wins 176 are draw and 0 where the second opponent wins. This is a significant reduction over the 255, 168 possible games with a complete game tree. These reductions can be very useful for devices with limited computing capabilities but also we prove a very important point that planning can be very efficient if designing meaningful game trees by applying reasoning very similar to human player reasoning.

Further improvements to the game tree are also possible if the opponents moves are also planned, in other words if we drop all the meaningless and symmetrical moves of the opponent.

6. Game AI in Monopoly

6.1 Overview of the AI Implementation

The AI agent is responsible for the moves of the artificial players in the game. The core principle of the AI agent is building a Game Tree with all the sensible moves that all the players would make from the current point of time forward. Then using the minimax algorithm the agent selects the move that in the future would bring the computer player most favorable game position with the highest probability. Building a Game Tree in this game that would be big enough to consider sufficient number of moves is obstructed by the vastness of possible moves in combination with all the possible random landings of the dice. The number of nodes of the game tree exponentially grows at each level. To tackle this problem the AI agents incorporates two already discussed technologies: Case Based Reasoning and AI Planning.

The technologies are employed in the following manner. First the agent searches the CBR database to find the case with the largest similarity with the current state of the board. This case is associated with a playing strategy. The strategy consists of goal that the planner needs to build plans for, and the plans consist of consecutive player moves that bring the player to that goal. This way only moves that are part of that strategy are considered, those being a small fraction of the overall possible moves the number of edges of the game tree at each level decreases immensely.

At each level of the game tree the model considers the moves of a single player. After the strategies of the AI player are considered the response to those strategies needs to be considered by the opponent(s). The move of the opponent(s) depends of the probability distribution of the dice as well as the strategy of the player. A more general strategy needs to be implemented for the opponent's (human player) moves since we cannot be aware of the expertise of the opponent. This general strategy would bring more plausible moves than the focused strategy of the AI player.

After covering all opponents the agent comes back to deducting a feature move of the computer player by using the CBR selected plan strategy. After creating several loops of strategies and reaching a reasonable size of a Game Tree taking into account the memory limits and the rapidly decreasing probabilities that the move is possible due to the distribution of the dice the building of the Game Tree stops. Then the minimax algorithm searches the Game Tree and decides on the most favorable move for the AI player using the minimax algorithm. The process is repeated each time the AI player is up.

Buying, auctioning and trading game moves are always accompanied by return of investment calculations in making the plans. These calculations represent adaptation of the more general planning associated with the cases in the CBR database. These adaptations are necessary due to the fact that the cases do not identically correspond to the situation on the table. In addition calculating the game position value of each node of the game tree is done by heuristic functions that incorporate economic calculations of net present value, cash, and strategic layout and so on. For example railroads in monopoly are known to be strategically effective because they bring constant income even though the income can be smaller than building on other properties.

6.2 Details on the CBR Implementation

The implementation of the CBR is by using the JColibri2 platform. JColibri2 is an object-oriented framework in Java for building CBR systems that is an evolution of previous work on knowledge intensive CBR [14].

For this implementation we need to look into three particular classes of the JColibri2 platform. The StandardCBRApplication, Connector, CBRQuery. For a JColibri2 implementation the StandardCBRApplication interface needs to be implemented.

The CBR cycle executed accepts an instance of CBRQuery. This class represents a CBR query to the CBR database. The description component (instance of CaseComponent) represents the description of the case that will be looked up in the database. All cases and case solutions are implementing the CaseComponent interface.

The JColibri2 platform connects to the CBR database via a Connector class. Each connector implements all the necessary methods for accessing the database, retrieval of cases, storing and deletion of cases. This implementation uses a custom XML structure for holding the CBR cases. Since the game will not update the CBR database only read it, a XML solution satisfies the needs. The XML file to a certain extent is similar to the XML representation of the board. We are interested in finding one CBRCase that is the most similar case to the situation in the game at the time of the search. This procedure is done in the cycle method of the CBRApplication. The JColibri2 CBR comparison is done by Nearest Neighbor (NN) search method.

JColibri2 offers implementations for NN search algorithms of simple attributes. These implementations are called local similarities. For complex attributes like in our case global customized similarity mechanisms need to be implemented.

The MonopolyDescription class [Figure 13] is basically a serialization of the GameState. It holds all the information about the state of the board, the players, their amount of cash etc.

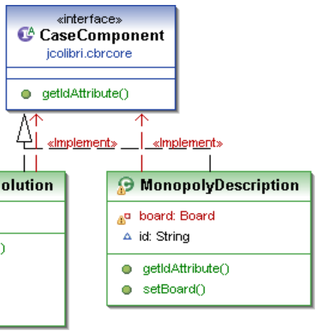

Figure 13: Class diagram of the Monopoly Case component models

On the other hand the MonopolySolution class holds the three particular attributes that are needed for the planning, the planning Domain, State and TaskList.

The game is implemented by using the Model-View-Controller software development pattern. The controller is responsible for implementing the game rules and handling all of the events in the game like roll of dice, input commands for trading, auctioning and etc from the players. The View layer is responsible for displaying the board and all of the input widgets on to the game screen, and the models are data structures representing the game state [Figure 14].

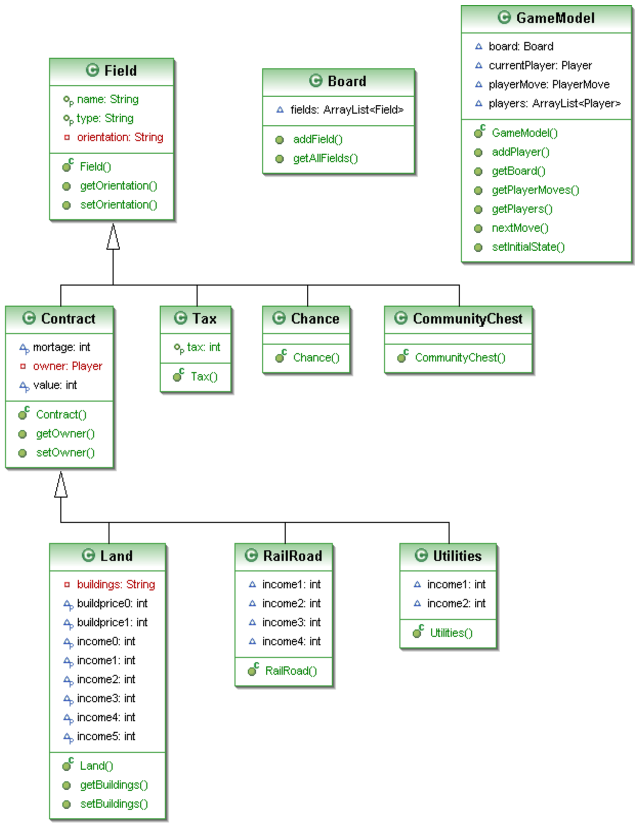

Figure 14: Class diagram of the Monopoly models

6.2.1 *Complex Similarity representation in CBR*

The similarity measurement part of the Nearest Neighbor algorithm JColibri2 is implemented by implementing the LocalSimiralrityFunction and the GlobalSimiralityFunction interface. A local similarity function is applied to simple attributes by the NN algorithm, and a global similarity function is applied to compound attributes. In the case of our implementation the attributes of the MonopolyDescription are compound attributes describing the state of the board, number of players, amount of cash for every player and etc. Since MonopolyDescription is a custom CaseComponent a global similarity function needs to be implemented to accurately find the distance between different CBR cases.

The similarity mechanism is inseparable core element of the CBR system. This mechanism represents how the CBR decides which strategy is best suited for the particular situation by calculating the distance or similarity to other cases in the database.

For the monopoly implementation we need to consider several basic strategies. Monopoly is based on investing in properties and receiving revenues from those investments. One of the basic strategies of the game is to build a set of properties that will bring constant income larger than the one of the opponents. So in time the opponents will have to declare bankruptcy. But on the other hand over investment can lead to too stretched resources with low income that will eventually drove the player to bankruptcy. To decide on these two we need a clear separation into two groups of cases in the CBR database. The first group of cases will represent a situation on the board where the player has significant income per loop formed of one or more color group properties, maybe railroads, some buildings on them and so on. It is important to note that in this case the player is better situated than his opponents so he only needs to survive long enough to win the game. In the other group of cases either the opponent is not well positioned on the board or its opponents are better situated. In this case further investments are necessary to improve the situation so the player can have a chance of winning in the long run.

These metrics can be owning color groups, valuing groups of railroads, evaluating the other opponents as well, and considering the amount of cash. As it is obvious in monopoly the number of streets is not as nearly as important as the combination of streets the player owns. It is also important to note that one CBR case does not hold only a single strategy in place, but its solution can have multiple different strategic goals. For example one CBR case might simultaneously say buy this land to form a color group but also trade some other unimportant property to increase cash amount.

The cases do not represent all possible combinations of board positions. They are only representation of typical game scenarios. The CBR Case solutions do not give exact instructions in general but rather strategic goals. For example one CBR Solution might say trade the streets that you only have one of each for the ones that you have two of that color already. Then the planner based on the situation on the board needs to decompose this high level task to a low level operations. Like offer "Mediterranean Avenue" for "Reading Railroad" and offer $50. The exact amounts and actual streets are left to the planer to evaluate.

The monopoly CBR database is currently in development on a monopoly clone game called Spaceopoly. The cases are architected based on human player experience and knowledge. There is a plan of making a number of slightly different strategies that differ on the style of playing and then running simulation tests that would determine the particular validity of each database as well as validity of certain segments of the strategy or even particular cases in the database.

The actual execution of the strategies will not differ from strategy to strategy since the plan execution is more related to the structure and rules of the game than to the actual playing strategy.

6.3 Details on the Planning Implementation

For the purpose of planning this implementation uses a modification of the JSHOP2 planner. The Java Simple Hierarchical Ordered Planner 2 is a domain independent HTN planning system [15].

JSHOP2 uses *ordered task decomposition* in reducing the HTN to list of primitive tasks which form the plans. An ordered task decomposition planner is an HTN planner that plans for tasks in the same order that they will be executed. This reduces the complexity of reasoning by removing a great deal of uncertainty about the world, which makes it easy to incorporate substantial expressive power into the planning algorithm. In addition to the usual HTN methods and operators, the planners can make use of axioms, can do mixed symbolic/numeric conditions, and can do external function calls.

In order for the JSHOP2 planer to generate plans it needs tree crucial components: Domain, State and Tasks. The Domain defines all the functionalities that the particular domain offers. These are simple and complex tasks. The complex tasks also called methods create the hierarchy with the fact that they can be evaluated by simple tasks of other complex tasks. This is how a hierarchical structure of tasks is formed. The problem reduction is done by reducing the high level complex tasks to simpler until all the tasks are primitive. The list of primitive tasks forms the plan.

The State represents the state of the system. It is a simple database of facts that represent the state of the system. The State is necessary to determine the way the problems or tasks are reduced to their primitive level. The reduction is done by satisfying different prerequisites set in the methods; these prerequisites are defined in the state. The Tasks are high level tasks or methods defined in the Domain. The planner based on the State and the goals selects one or more high level tasks that need to be reduced to plans [Figure 15].

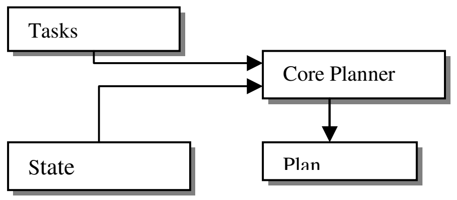

Figure 15: Diagram of a Planner

The plans then generate the game moves. The number of moves generated by the plans is just a fraction of the possible moves at that point. This reduces the game tree providing the opportunity to generate smaller and deeper game trees and making more efficient decisions in general.

7. Conclusion

Even though the results from the CBR database are not complete at this time partial strategies are implemented as cases and recognized during game play by the CBR system. These smaller local strategies coupled with more global higher level strategies that are particularly important at the beginning of the game would form a complete CBR database and represent a knowledge engineered style of playing of the AI player.

The AI Planning approach is a proven method by the tic-tac-toe experiment and is suitable for implementing the strategies associated with the CBR cases.

This approach in general benefits from both technologies, CBR as well as AI Planning and comprises an elegant solution. Even though AI Planning can be enough as a single technology for some simpler problems like tic-tac-toe the complexity of Monopoly would mean that the Planner would have to incorporate large and complex domain and a very big state model. The CBR application helps reduce this complexity by focusing the planning on smaller domain of the game. Basically the CBR reduces the overall goal of the play (wining the game) to smaller more concrete goals suitable to the particular state of the game, thus reducing the need for global planning strategies and complex planning domain.

Furthermore this symbiosis of technologies gives way for more precise and finely tuned strategies which can be difficult to include into global plan for the whole game. One simple example for the Monopoly game would be this: Sometimes it's better to stay in jail because rolling double increases the probability of landing on some field (two, four, six, eight, ten or twelve steps from the jail) that can be of great importance to the rest of the game. These and similar small local strategies can be easily recognized by similar cases in the CBR database.

In other words the system is flexible enough so that new strategies can be incorporated easily missing strategies can be also recognized by the distance metrics as well as wrong assumptions in the strategies can be easily recognized.

One other important property of the system is that is highly configurable. The game its self can be diversely different depending on the configuration of the board. Even though the platform is restricted to Monopoly type of games, changing the layout and values of the fields effectively brings completely different properties of the game. In addition the CBR database represents the entire experience of the AI Player. It can be filled with rich set of strategies or even configured with different flavors of difficulties of play, this of course coupled with the domain of the planner which can differ from a case to a case as well.

8. Future Work

Further exploration of this technology would go towards complete implementation of an AI aware agent for monopoly. Initial results from the local cases with more specific strategies show CBR as a capable tool for representing expertise in playing the game. Completing the more general strategies and coupling them with the planning domain will give precise results on the benefits from this architecture.

There is also need for exploring the planning of strategies of opponents. This task is to some extent different because we cannot always expect the opponent to select the best move we think. In the Tic-tac-toe example all possible moves of the opponent were taken into consideration, if we used the same planner for the opponent only tie games would result from the game tree. In other words mistakes of the players also need to be considered.

The CBR Platform brings other functionalities well worth of exploring as well. The revision stage of the JColibri2 platform is basically capable of fine tuning strategies or even developing new strategies for the games. A well written underlying AI planning model with a capable feedback of the game tree evaluation back to the CBR revision capability can be an interesting concept in automatic experience acquisition for the AI model.

There are also many other fields were combined CBR and planning approach can be incorporated into a problem solution. This combination is analogous in a big extent to a human way of reasoning. People in addition to logic of reasoning in situations with lack of information rely to planning strategies and prior experience, exactly the intuition behind CBR – AI Planning architecture.

9. ACKNOWLEDGMENTS

We would like to thank Prof. Sofia Tsekeridou for her involvement in the valuable discussions we had on the topic of CBR.

10. REFERENCES

[1] Minimax. *Wikipedia.* [Online] [Cited: April 23, 2008.] http://en.wikipedia.org/wiki/Minimax.

[2] **Von Neumann, J**: *Zur theorie der gesellschaftsspiele* Math. Annalen. **100** (1928) 295-320

[3] Automated Planning. *Wikipedia.* [Online] [Cited: April 23, 2008.] http://en.wikipedia.org/wiki/Automated_planning.

[4] **Sanchez-Ruiz, Antonio, et al.** *Game AI for a Turn-based Strategy Game with Plan Adaptation and Ontology-based retrieval***.**

[5] **K. Erol, J. Hendler, and D. Nau** (1994). Semantics for hierarchical task-network planning. Technical Report TR-94-31, UMIACS.

[6] **Smith, S. J. J. and Dana S. Nau, T. A. Throp.** A Planning approach decrarer play in contract bridge. *Computational Intelligence.* 1996, Vol. 12, 1.

[7] *One Jump Ahead: Challenging Human Supremacy in Checkers.* **J.Schaeffer.** s.l. : Springer-Verlag, 1997.

[8] **IBM.** How Deep Blue works. [Online] 1997. [Cited: April 23, 2008.] http://www.research.ibm.com/deepblue/meet/html/d.3.2.html

[9] **Ghallab, Malik, Nau, Dana and Traverso, Paolo.** *Automated Planning theory and practice.* s.l. : Morgan Kaufmann Publishers, May 2004. ISBN 1-55860-856-7.

[10] *Case Based Reasoning. Experiences, Lessons and Future.* **Leake, David.** s.l. : AAAI Press. MIT Press., 1997.

[11] *Applying case-based reasoning: techniques for enterprise systems.* **Watson, I.** San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1998.

[12] **Plaza, A. Aamodt and E.** Case-based reasoning: Foundational issues, methodological. *AI Communications.* 1994, 7(i).

[13] Tic-tac-toe. *Wikipedia.* [Online] [Cited: April 23, 2008.] http://en.wikipedia.org/wiki/Tic-tac-toe.

[14] **Díaz-Agudo, B. and González-Calero, P. A.** An architecture for knowledge intensive CBR systems. *Advances in Case-Based Reasoning – (EWCBR'00).* New York : Springer-Verlag, Berlin Heidelberg, 2000.

[15] **Ilghami, Okhtay and Nau, Dana S.** *A General Approach to Synthesize Problem-Specific Planners.* 2003.