MAN – MACHINE INTERFACE

S.Bhuvaneswari, Reader, Pondicherry University, Karaikal Campus, Karaikal,
e-mail: booni_67@yahoo.co.in
R.Hemachandran, Faculty in Mechanical Engineering, N.I.T., Puducherry
Suman Kumar Pandey, Scholar, Pondicherry University, Karalkal Campus, Karaikal

ABSTRACT

Agents trained by learning techniques provide a powerful approximation of state spaces in games that are too large for naive approaches. In the study Genetic Algorithms and Manual Interface was implemented and used to train agents for the board game LUDO. The state space of LUDO is generalized to a small set and encoded to suit the different techniques. The impact of variables and tactics applied in training are determined. Agents based on the techniques performed satisfactory against a baseline finite agent, and a Genetic Algorithm based agent performed satisfactory against competitors from the course. Better state space representations will improve the success of learning based agents.

KEYWORDS

AI, State spaces, GA, Intelligent Agents

1. Introduction

Artificial Intelligence Ludo (ai-ludo) is an environment for artificial agents. Its purpose is to compare several approaches in the area of artificial intelligence. On the one hand, Ludo is rather a simple game and fully observable but, on the other hand contains a few challenges due to the stochastic and multi agent environment. Therefore it offers a good balance between simplicity and complexity and is able to attract a wide audience and not only professionals. In addition, it is very common and well known around the world. The goal was to provide a simple platform on which everybody could add an own player and check its strength compared to several other ones. By providing a possibility to create game statistics it is feasible to analyze the various players in detail. As first players we implemented the Manual Player and the genetic learning player. The genetic learning player bases decision on a utility function to evaluate all possible following game states. In this case, the utility function is composed of several terms and their weights are determined beforehand with the help of a genetic algorithm. We will show that the approach based on a genetic algorithm is applicable and profitable and significantly outnumbers the player in wins.

First this paper introduces the ludo game rules players must adhere to be comparable with other players. Afterwards the theory about utility functions and genetic algorithms as well as the concrete application on the Ludo game is described. In the end, our experimental results as well as their evaluation are presented.
This study concerns the use of learning based agents in the game LUDO. The LUDO game is rather complex systems that contain dice which adds randomness as a factor. It is implemented in a JAVA framework with a number of finite agents to be used for training against. It is difficult to

model the game using classical AI techniques, which require a complete model of the state space. This includes constraint based agents, adversarial search based agents and logic based agents, albeit probability theory can be used for these to predict the random outcomes. The state space is the second difficult factor, because it is enormous as there are 16 bricks in total divided among four players, and 52 configurations for each brick, resulting in a full state space of 1652. This makes global search practically impossible, at least for this study.

Instead local search based learning techniques are used, which do not require a complete model of the state space. Rather they in different ways approximate this space through learning and adaptation. Three techniques are investigated: Genetic Algorithms (GA). *Purpose* Implement experiment and compare the performance of learning agents in the game LUDO, using Genetic Algorithm.

2. Purpose

Implement Genetic algorithm based agent and a Manual player environment which is Human interface to chess agent the Genetic Learning agent. For experiment and compare the performance of learning agent and Manual player provide a LUDO game simulator. After a number of simultaneously played games show the performance result of the each agent and deduce the performance ability among them which one is best for learning technique.

3. Game Rules

There are several ludo games available. The game consists of a board with 40 fields, 1 begin field, 4 start positions, 4 pawns and 4 end fields for each player.

A pawn is moveable if the dice count added to the pawn's current position would not lead to a field which is already possessed by another player's pawn and it would not lead to hypothetical field which is behind the end fields. The game is won by a player, if he is the first to have all his pawns on his end fields. The rules of this version are as follows:

• Any player who throws a 6 has to take an own pawn from the start position to a begin field.
– If there is no pawn left on the start position, the player is allowed to move
any moveable pawn.

• Afterwards, the player is allowed to throw the dice again and to move the according number of points.
– The begin field has to be freed as soon as possible.
– If there is no pawn left on the start position the pawn is allowed to stay on its begin field.

• If a player's pawn A is moved to a field which is possessed by an opponent's pawn B the opponent's pawn is thrown. This means, pawn B has to be moved back to the corresponding start position.
– If the field is possessed by another player's pawn C, the player is not allowed to move the

• If there are several pawns of a player on the board, the player is allowed pawn A to this field to choose which pawn to move.
– Only one pawn per player is allowed to be moved in one round.
• If a player has no pawns on the field he has 3 attempts per round to throw the necessary 6.

4. Related Work

From the first stirrings in the area of artificial intelligence until today, games have always taken an important position. Often it is easier to test new algorithms on closed environments with a fixed set of rules, like games, before applying them on real world problems. In many cases the first idea is to use search algorithms in order to find a solution, for example like Deep Blue1 the chess playing agent does it. If a search cannot be used because of the large search space or space/time restrictions, heuristic methods, like the A*-Search or genetic algorithms, may provide a basis for building artificial players. Beside the search strategies, a second huge class of algorithms is concerned with learning. These ones, like neutral networks reinforcement learning, require some training data and compute the best moves for an artificial player beforehand and not directly during the game. For nearly every imaginable game artificial players have been developed based on at least one of the two basic methods.

In addition, for most of them exists some competitions to prove which player is the best one. Even for the Ludo game there is a huge amount of artificial player implementations included in almost every Ludo game is playable on a computer. Unfortunately most of them do not provide any information about their internals, but it would be nice to compare the technique for Ludo implementations.

5. STATE SPACE

A smaller state space is necessary for the learning techniques to converge in realistic time, albeit the information is less complete than the full state space. This makes the design and choice of the state space vital for the success of the agent, no matter the duration and parameters of training.

A full state space contains within it the optimal solution, along with countless local solutions that are suboptimal. It is theoretically possible to use learning on the full state space, but even for a game like LUDO, this space is enormous. Given enough computing power, time and space, it would however be possible to learn this complete space, and
the optimal solution within.

All the implemented learning techniques share the same state space. This state space is partly derived from the provided LUDO agent as well as additional information about the game. Each state is the consequence of moving a brick with the given dice.

The states are the following in the given order:--
1. Brick can hit an opponent home.
2. Brick can enter the goal.
3. Brick can hit itself home.
4. Brick can hit a star.
5. Brick can enter a dangerous field.
◦ A dangerous field is a field in which opponents can hit the brick home in the near future.
6. Brick can enter a safe field.
◦ Fields such as globes, the last fields leading to the goal or if the field is not dangerous.
7. Brick can move out of start area.

The representation of states varies for each technique, GA assign weights to states .Despite the difference in representation, the states can be interchanged between techniques, allowing hybrid approaches to readily be applied.

6. Background Knowledge

In our project, we developed an artificial player which internally uses a utility function. The weights of this utility function are computed using a genetic algorithm.

To get a first impression about utility functions and the functionality of genetic algorithms in general, the following section describes the theory behind it.

Utility Function

Utility functions are often adapted within the construction of artificial agents. They are used to internally evaluate the states of the environment. This can also be helpful to estimate the utility of possible following games states in the case the agents performs a certain action. How to model the utility function is very problem dependent and has to be considered on a case-by-case basis.

As counterpart to the utility function, the performance measure is often used to evaluate how well agents operate but their behavior is observed from the outside. Due to the different point of views on the agents' performance, they might result different values and thus should be distinguished.

Genetic Algorithm

The genetic algorithm is a search heuristic which can especially be used to find solutions to optimization and search problems where the search space is too huge to use traditional methods. Based on the fact that it is only a heuristic, the found solution must not necessarily be the optimal one, but in most cases it works quite well. In general, the idea is to imitate the biological evolution. The whole process is depicted in the following Figure and has to take several steps.

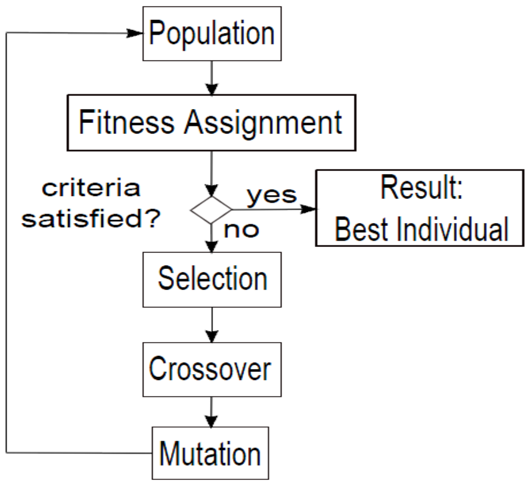

Fig.1. Process of a genetic algorithm

First, a population is needed which consists of several individuals whereby each individual has a number of genes. In addition, a fitness function is required which assigns every individual a certain value. According to this values, the individuals can be ordered and it is possible to determine the best individual among them. If a specified termination criterion is satisfied, for example the value of the best individual is above a threshold or 20 generations have been generated, the algorithm terminates and returns the best individual. Otherwise, the individuals with the lowest values of the fitness function are deleted from the population. Afterwards, a new generation arises by crossover whereby the new individuals have genes compounded of the genes of their parents. In addition, mutation of genes is also possible whereby some genes are changed randomly to ensure genetic diversity. After creating the next generation, the whole procedure starts again with the new population.

Only the fittest individuals of a population survive and thus a new generation consists of individuals with genes which have already been proved as good ones based on the fitness function. This implies that a very late generation is just made up of individuals with the best genes. To solve problems with this approach, it is first required to model the individuals with their genes and a according fitness function. Usually, the genes are potential solutions, for example assume an optimization problem compounded of a number of variables and for each variable a value should be assigned such that these variables solve the problem as good as possible. In this case, it can be modeled that a gene corresponds to exactly one variable. Then a fitness function might assign the score of the variables, which is reached when applied to the optimization problem, to the individual. Furthermore, it is also necessary to define how many individuals to delete, the termination criteria as well as a strategy for crossover, e.g. how to determine the parents, and one for mutation, e.g. how many genes of how many individuals should be mutated. In the end, the best individual is returned, which represents the best possible solution the algorithm detected.

7. Player

There are two types of artificial players implemented: a Manual player and an evolution player (also called genetic learning player). The strategy of the genetic learning player is described in the following section.

Genetic Learning Player

After explaining the theory of utility function and genetic algorithm we can make use of it and adapt it to our idea of implementing an artificial player for the Ludo game. Now in this section, the implementation of the genetic learning player is described.

Utility Function:

Internally, the genetic learning player is based on a utility function assigning a score to each possible resulting game state. This score is a sum of several positive and negative terms describing different situations.

Following situations should result in a positive term:--

• Directly beat a pawn of another player.
• Get into a position where another pawn is potentially beatable in the next round.

• Get out of a position where an own pawn is potentially beatable by another player.
• Get pawn on an end field.
• Get as close as possible to the end fields.

Following situations should result in a negative term:--

• Move own pawn into a position where it can potentially be beaten by Another's player pawn.
• Move pawn on a begin field of another player.

Potentially beatable means that the distance between two opponent pawns is less or equal to six. We also considered other situations, but we concentrated on these ones because the utility function should be as easy as possible and nevertheless simultaneously cover all important cases. After defining the utility function, it is necessary to determine the values of the terms in detail. Therefore we used a genetic algorithm to provide best possible values.

8. Application of a Genetic Algorithm

*I*t is necessary to model the individuals as well as several strategies, e.g. for mutation, to adapt the algorithm to this problem. In our case, the genes' of an individual are the utility function terms and they can have integer numbers between 0 and 100. To evaluate the genes we used a fitness function which just counts the number of victories an individual achieves in 100 games against a random player, e.g. 80 of 100 won games results in a fitness assignment of 80. Our starting population consists of individuals with randomly generated genes. Tests with different termination criteria together with different mutation rate and number of deleted individuals have been performed and will be presented later in this section. In general the parents, the composition of the genes for new individuals and the genes to mutate are randomly chosen. It is assumed that after a while the best individual will have approximately optimal genes and therefore the perfect terms of the utility function are found.

The whole process is depicted in pseudo code in Algorithm:

```
Algrorithm GA(populationSize)
{
for i=1 TO populationSize do
generate(i,random)
score(i) = wins out of 100 games
population add(i)
end for
while !terminationCriteria do
{selection}
for i=1 TO numberOfNewIndividuals do
population delete(weakest)
end for
for i=1 TO numberOfNewIndividuals do
{crossover}
parents(i) = random individuals i1, i2
breakingPoint = random(1,7)
generate(i,i1,bp,i2)
population add(i)
{mutation}
if mutate(i) then
mutationPosition = random(1,7)
mutationValue = random(0,100)
mutate(i,mutationPosition,mutationValue)
end if
{fitness assignment}
score(i) =  wins of 100 games
end for
end while
return population get(best)
}
```

In the first for-loop, the starting population is generated containing individuals with random genes. Afterwards the while-loop is repeated until the termination criteria are fulfilled. Now the selection, crossover and mutation steps are carried out. During the selection the individuals which lost the most games against a random player are deleted of the population. Afterwards the new individuals are generated whereby the parents i1 and i2 are selected randomly as well as the breaking point. The child gets all genes from i1 till the breaking point and the other ones from i2. If enough individuals have been created, some of them are mutated if mutate (i) is evaluated to true whereby the decision only dependents on the mutation rate. Again, the gene to mutate as well as its new value will be randomly chosen. In the end, each new individual plays 100 games against a Genetic Algorithm player to determine its fitness score and the whole process starts again.

9. Experiment

If the algorithm terminates after 10 generations, it is not really obvious whether a high or a low mutation rate is better. After 25 generations, it becomes clearer that either only a few individuals should be mutated or all new ones. This is based on the fact that on one hand good individuals might be mutated and get worse but on the other hand it prevents the population to head a set of genes too early which might not be the best possible ones. It is also reasonable that a termination criteria of 10 generation is too low because the genes might still be widespread. Thus for all following tests we decided to set the mutation rate to 1.0% and the algorithm terminates after 25 generations have been created. Afterwards we changed the number of individuals which are deleted or conversely the number of new individuals which are created. If only a few individuals are generated, many individuals with rather bad genes stay in the population and in addition new individuals might be created consisting of these genes.

Therefore we fixed the values 0.75 for the rate of new individuals which are created per generation. At the end, we tested how changes in the population size affect the outcome. Except for some small deviations, the size of the population does not significantly change the result.

To get in the situation that it might be possible to beat the opponent in the next round as well as to escape a possible beat is rather not such important. This is of course not a general rule and might change if the opponent has another strategy than just randomly selecting a pawn. All other values are not really significant and are also partially different when using various sizes of populations.

population size	100	250	500
directly beat opponent	95	73	91
possible beat next round	4	11	15
escape possible beat	0	9	0
enter beat range	50	81	57
get on a start position	3	35	44
enter an end field	7	73	37
distance to end fields	55	27	25
best score	95	94	94

Fig 2. : Term values for different populations sizes

10. Implementation of GA

The GA technique is implemented in a JAVA package, consisting of a population of chromosomes encapsulated in a GA class. For convenience, the implementation is general purpose, and can thus be used in other problems than just LUDO. The chromosomes use floating point numbers for their genes in the range [1;0], allowing direct mapping to the state space in the form of weights. Chromosomes start with random values within the range.

The population size is fixed at an amount of 100. Selection is based on the elitism approach, where the ten chromosomes with the highest fitness value are kept. All reproduction combinations save with themselves are performed in order to create 90 children, thus keeping the population at a fixed size of a 100. The reproduction strategy is cross breeding, such that even numbered genes are of the first parent, and odd numbered genes are of the second parent. Mutation is done through random selection of a gene, and adding a random value in the range [-0.5;0.5], while maintaining the bounds of the range.

The agent utilizing GA contains a single chromosome to weigh the consequences a brick, and the brick with the highest value are then selected. Eight genes are used; one more than the state space, as all moves that is not described through the state space representation is weighted as an eighth. This is necessary because weights are in the range [0;1], and bad moves must be able to be marked lower than a move without benefits. Optimally, good moves are thus selected and bad moves are refrained from after training.

Each chromosome within the population competes in a number of games, and the number of victories is accumulated as the fitness score. Since fitness is determined by wins alone, there is no need for a special agent to perform the training, and the GA agent is used by swapping its chromosome throughout the tournament. This procedure is continued for a number of generations, after which the chromosome with the highest fitness value are saved and used for the final agent.

11. Conclusion

Summing things up, the goal of this project was to compare several approaches in the area of artificial intelligence under the constrained environment of a Ludo game.

We, therefore, implemented a Manual player and a Genetic learning player who uses an adaptable utility function to determine the most useful next action. The results are pretty clear: The Manual player was beaten by the genetic learning player win with its best configuration in over 80% of the time. So, the main question is answered: The genetic learning player is able to learn from situation like the Manual player.

One might argue to add or remove criteria from the utility function and see how the genetic learning player performs under these new constraints. It would even be possible to compare two genetic learning players each trained with a different utility function. Further, more artificial players might be added to the application.

Scope: There is also another learning technique which I can use as my extended project, like Neural Network Architecture (ANN) and Reinforcement-learning technique instead of Manual Interface. It can be extended to a network based application in near future also.

Limitation : For making comparison Human beings are not capable to give appropriate Learning Environment for LUDO. So it is necessary to implement another approaches. As LUDO board is developed to consider to run in standalone system. It is also dependent for System monitor view port.

References

1.Artificial Intelligence , "A Mordern Approch", second edition, Stuart Russel and Peter Norving
2. Neural Networks, Fuzzy Logic, and Genetic Algorithm , " Synthesis and Application",
S. Rajasekaran and G.A. Vijayalakshmi Pai