FOUNDATION OF ARTIFICIAL INTELLIGENCE

Vampire vs Werewolves game

Author
Alexandre Robin
Olav de Clerck
Pierre Nikitits

Contents

Introduction

The aim of this project is to play the Vampire vs Werewolves game. In this game two faction, Vampire and Werewolves are opposed and need to fight to win. They can use the people of a third faction to enlarge their ranks by converting them, the humans.

The appeal and complexity of the game lies in the trade-off between attacking the enemy faction and attacking humans. We will try to develop an artificial intelligence that is able to play and win the game.

1 Modeling of the game

In our project the game map will only be represented by an enumeration of occupied locations. In other words, our map modeling is done in the form of a sparse matrix. It will allow us to save space and time during the computation of moves.

The matrix consists of three lists - one per faction - of objects that represent the positions of the allied groups, the human groups, and enemy groups. Each object includes the position of the group and the number of individuals in the following format:

Place = [x, y, #(humans), #(vampires), #(werewolves)]

After each turn the matrix is updated to describe the current game plan. This modeling allows to reduce the calculation time required for each game board update and to free up maximum computing power for the algorithm that will choose the best moves to play.

2 Playing strategy

Given that the game involves two players and the opponent's actions are not predictable in advance, it is necessary to develop an algorithm capable of predicting the opponent's moves while selecting the best possible move to ensure the optimal mid-term situation. With this in mind, we initially chose to implement an enhanced version of the minimax algorithm, incorporating the alpha-beta pruning technique.

In this algorithm, we successively examine the various possibilities of moves, conducting a thorough exploration of the tree of situations for each move. We continue this exploration until reaching a defined depth, where we evaluate the game situation using a heuristic function detailed in section 3. The heuristic values calculated at the limit depth are then retroactively propagated to the root, allowing us to select the best move to execute.

The efficiency achieved through pruning will be exploited to further extend the depth of the tree. When the server's turn is given, our Minimax algorithm begins by extracting the current state of the board, then creates a copy to simulate the outcome of each potential move and the subsequent moves.

At the end of this process, we identify the best move to play, which we then transmit to the server.

Once the structure of the alpha-beta algorithm is in place, it is the heuristic that is decisive in determining the strategic capabilities of our AI.

3 Heuristics

Due to the high combinatorial complexity of the game, it is impossible to explore the entire decision tree. Therefore, it is imperative to restrict the search depth of our artificial intelligence. In this context, it becomes essential to choose an effective heuristic capable of evaluating each situation on the map accurately, thus reflecting the chances of victory. We therefore seek a heuristic that can differentiate between two possible moves at any moment in the game for any game state.

First remark

During the construction of the algorithm, as stated earlier an important parameter is the depth of the tree that we build. The depth corresponds to the number of moves the algorithm will consider when searching for the best move. Based on the understanding of the algorithm and the depth parameter the first important thing to note is the following:

- Considering the local environment around an ally is not a viable strategy.

Indeed, supposing a situation where the neighborhood of an allied group with radius corresponding to the depth of the algorithm is empty, it will be impossible to derive a move. The conclusion of this remark is that the heuristic evaluating a position should not only consider a close neighborhood byt also consider element of the map wherever they are, from which follows:

- A component of the heuristic should be a distance rather than descriptive statistics.

Considering a distance will not be a local metric and work whatever the positions on the map.

Determining metrics

Win. Obviously the most important component of the heuristic is the win flag. Whenever the exploration during the construction of the move tree shows a win situation it should select it.

Population. Another obvious metric of the heuristic should be the difference in population between the allied groups and the enemy groups. Having the greatest numbers is a big advantage to win the game.

Distance to eatable human group. As stated we also need distances in the heuristic metrics. Supposing the neighborhood of an allied group is empty, the two metrics described above (Win and Population) will not yield any result. The best move can then be to get closer to a human group that we can eat in order to improve our numbers. We consider in this metric the closest human group.

Distance to eatable enemy group. Following the same reasoning as for the Distance to eatable human group metric, it can be interesting to get close to an enemy group that has inferior numbers. Not only because we can eat them and win, but also because we could perturb their decision making. We consider in this metric the closest enemy group.

Distance to threat. Another interesting distance is the Distance to threat, it corresponds to the distance between an allied group and a superior enemy group. When getting toà close to this group, the best move can be running away. However it will be important to not overweight this metric to not perturb our play. We consider in this metric the closest dangerous enemy group.

Combining the metrics

We then need to combine the metrics and attribute weights in order to have a functional heuristic. In this section we based our search of good weights on the course of Decision Modeling from Vincent Mousseau. We consider our heuristic to be a preference model for which we have 5 metrics and we want to find the best weights. We use a method of elicitation of the importance of the coefficients and consider an additive model. Despite being driven by the concepts of decision modeling in the big lines, our search remains empirical.

We start by defining situation and ranking them we will then derive values for the weights.

First note that since we want to minimize the distance to eatable groups we will use the inverse value of the distance for these metrics.

Win situation. A win situation should be chosen anytime. The associated weight need to be able to offset any combination of the other weighted metrics. Since a win is a binary variable represented by 0 or 1, we will consider a very big weight for this metric.

Eat human or enemy group. In a situation where there are two identical groups of human and enemy, at the same distance from an allied group, we will consider that it is better to eat humans rather than enemies, because the enemy group can move. The weight of the Distance to eatable human metric should then be bigger than the weight of the Distance to eatable enemy metric.

Eating situation. When getting near a group that can be eaten, we want to favour the kill rather than the proximity to the target. It translates in a bigger weight for the Population metric than for the Distance metric. A Human at distance 1 will have a Distance metric of $1/1 = 1$ and the Population metric will be improved of 1. We see that if the weight of the Population metric is not bigger, the allied group would go for the kill.

Running away situation. In a situation where the threat is very far away it should have the relative impact of the metric in the heuristic should be very small whereas it should be bigger if the threat is close.

Since we consider the inverse for several distance metrics we note that the max distance on the map is $1/15 = 0.06$. We attribute a weight of 1000 to get a round 60 in this case. In the case of the threat we consider the distance to improve the max is 15 and the min is 1. We give it a smaller weight so that is doesn' have too much impact .

The weights we considered are:

- 1000000 for the Win metric

- 10000 for the Population metric
- 1000 for the Distance to eatable human metric
- 800 for the Distance to eatable enemy metric
- 300 for the Distance to enemy metric

This choice of weights doesn't yield impressive results. An improvement could have been to not consider linear relationships between metrics. The Distance to threat metric could be considered in an exponential or polynomial form to be more interesting.

4 Quick look at Reinforcement Learning

During our work some members of the team remarked that Reinforcement Learning could be tried. Despite not being implemented, the following start of reflexion was made. The idea was to model observation and action spaces and to think of which kind of RL algorithm could be implemented.

Observation space. The observation space correspond to the current map. In order to have a fixed size, not varying with the game, we decided to consider only a part of the map. We consider as the observation space, for a given allied group the list of the 4 nearest allied group (making a list of 5 allied group considering itself), the list of the 5 nearest human groups and the list of the 5 nearest enemy groups. Since a group is described by 5 values, the number of parameters in the observation space sums up to $3 \times 5 \times 5 = 75$. If there is less than 5 groups in a faction the values are set to -1. To tackle the variability of maps in terms of size we would change x and y coordinate of a group to x/lengt and y/height. Making the observation space continuous and the values of coordinates ranging from 0 to 1.

Action space. The action space corresponds to the set of actions available for an allied group. There is 8 move available that we combined with 4 split possibilities (0%, 25%, 50% and 100%) summing up to 32 actions possible.

Reward. The reward we thought of was a basic 1 for win and -1 for loss.

Algorithm. The algorithm we thought could work for the game is the Reinforce algorithm as described in the gymnasium website. This algorithm calculates a policy, that is a mapping between the observation space and the action space. An epoch of learning would be a game and a step would be be a move.

IN order to train efficiently the agent we would ahve started with small maps (10×10) against our alpha-beta algorithm then we would have make the agent fight against itself.

5 Reflex agent

Facing the complexity of calibrating a good heuristic we tried to create a reflex agent in order to see what would be winning moves and so that we had a worthy opponent. The reflex agent was much better than any of the heuristic we found.

A simple algorithm that can play the game, in a predictable way is given by:

1. For each of our groups, we iterate over all humans and all enemies and compute the expected outcome if a battle were to occur, taking into account the battle probabilities.
2. We then compute a value as
$$\frac{\text{expected outcome}}{\text{distance}}$$
and order the targets by this value.
3. We don't consider targets with 100% chance of losing the fight, and if those are enemies (ie they can move) we create a 1 move buffer zone to avoid.
4. To include the splitting mechanism, we check if our group has more members than needed to 100% win against its target, and if so we only send the minimum number to win the battle.

Despite its efficiency and ability to split, this algorithm has certain limitations: it doesn't take into account the enemy's strategy, its strategy looks at 1 target at a time per group.

6 Limits of alpha-beta algorithm

1. Time: highly demanding in terms of computation.

The need for the alpha-beta algorithm to test the entire game tree before making a decision makes it a very time-consuming solution. In some situations, such as blitz chess, the calculation time can be crucial, making it relevant to explore techniques to accelerate the computations.

The first approach would be to leverage parallel computing capabilities to process multiple branches of the game tree simultaneously. This could significantly accelerate the search process. The second approach would be to explore the integration of machine learning into the model, justifying our consideration of reinforcement learning.

2. Heuristics randomized search:

Although derived from intuition, the heuristics we presented are primarily obtained through trial and error. To find the best possible heuristics, it would be relevant to test a large number of possibilities to identify the combination of heuristics that yield the best gameplay results.

3. Dynamic heuristics:

The heuristics we proposed are static, meaning they do not change throughout the game. However, depending on the state of the game and the progress of the match, certain characteristics of the board may have varying importance. It would be pertinent to implement dynamic heuristics that change during the course of the game.

4. Splits and merge in the alpha-beta:

Our alpha-beta does not consider the possibility of splitting and merging our creatures. However, during the tournament, we observed that this ability proved decisive in achieving

victory. Considering the possibility of splitting would, therefore, be a significant competitive advantage worth considering.