Vampires VS Werewolves
Artificial intelligence project

Jeremi Assael
CentraleSupélec
jeremi.assael@supelec.fr

Lorenzo Gasparollo
CentraleSupélec
lorenzo.gasparollo@student.ecp.fr

Manal Saadi
CentraleSupélec
manal.saadi@supelec.fr

March 31, 2019

1 Introduction/Motivation:

This report describes the deliverable of the project related to the course Foundation of Artificial Intelligence, in the Msc "Artificial Intelligence", under the supervision of professor Jean-Philippe Poli. The goal is to build an Artificial Intelligence to play the Vampire VS Werewolves game. The report starts with a general definition of the game, followed by an explanation of our strategy. After, we describe the code and python algorithms we used. Finally, we conclude with some remarks and improvement ideas.

2 Definition of the game

2.1 General Context

In a far far away land, mortal creatures lived a peaceful life. At nightfall, they helplessly witness a fierce struggle between two supernatural species: Vampires and Werewolves. Each night, humans have to stay in their houses. We are randomly affected to one of the two species and our goal is to be the dominant specie. To increase the number of creatures of our specie, we can change humans into your own species: a vampire's bite changes humans into vampires, a werewolf's scratch turns them into werewolves. To change a group of humans, the creatures , vampires and werewolf, have to be at least as numerous as them. The creatures can kill each other: they have to be 1.5 time more than their enemies. The game ends at daylight (after the game time ends) or when one of the species totally disappears the winner is the dominant species at that time.

2.2 Initial representation

The universe is a nxm grid. Some cases are humans' houses, they stay there and don't move. We can see all the creatures and humans in the game and their location: the universe is totally observable.

2.3 Rules

The game contains several rules that need to be respected, otherwise, the team not respecting them get disqualified and the other one wins automatically the party.

- At least one movement by ply

- We can only move your specie groups
- We need to have enough creatures on a cell to perform all the moves from this cell
- We can move in 8 directions (unless on borders)
- In the same ply, a cell cannot be target and source of moves
- We have to move at least one creature

2.4 Battles

In the case the player moves to a cell containing enemies or humans . Two types of battles are possible : Deterministic battles : battles with humans where, if the number of humans is lower than the number of creatures, the humans get converted to the same specie as the fighting creature, otherwise the creatures are killed. Battle with creature where, if we have 1.5 more members than them, we kill them, if they have 1.5 more members than us, they kill us. Otherwise it's considered a probabilistic battle.

Probabilistic battles : The end of the battle is given by the following probabilities. Let E1 and E2 be the number of the two species (opponents or humans) and E1 be the number of monsters attacking

- If E1=E2, then P=0,5 ;
- If E1¡E2, then P is given by the linear relation passing through (0 ;0) and (E2 ;0,5), thus P =E1E2 ;
- Otherwise P is given by the linear relation passing through (CxE2 ;1) and (E2 ;0,5) thus P = E1E2-0.5

The probability that E1 wins is given by P. However, the struggle has been tough and some of the attacking monsters died:

- If the attackers win, each of them has a probability P to stay alive. Moreover, if they win over humans, each human has a probability P to survive and to be converted;
- If the attackers loose, each enemy has a probability 1-P to stay alive.

3 Strategy

3.1 Alpha-beta

We decide to choose alpha-beta pruning using the min-max algorithm to find the best move to make at each step of the game after calculating the Heuristic needed in our decision process.We choose it because it returns the same logical moves as minimax would, but prunes away branches that cannot possibly influence the final decision. Which saves us time and space complexity and unnecessary calculations. We applied Alpha-Beta with depth 2 and depth 4 and kept for the tournament the alpha-beta with depth 4 because it is more efficient, takes in consideration deeper and more sophisticated information of different precedent states, and manages to perform all that in a reasonable and satisfying period of time thanks to the coding structure that we used.

3.2 Heuristic

In this part, we calculate the heuristic that we try to maximize in our decisions for the general strategy of the game.

3.2.1 Inputs and parameters

In order to calculate the heuristic, we take different element in consideration. We take as input our state and the state of the opponents: the lists of creatures and humans in the whole map (list of our creatures, list of the opponent creatures , list of humans), number of groups of humans and creatures. We get also constant data about the map : width, height and diagonal. In addition, we consider some numbers that we take as extremum : the maximum possible number of humans, inf which is a large value that we consider as infinity. According to our strategy, we add some parameters to our heuristic:

- **to-gh** is by default equal to 3, it represents the maximum number of good directions related to human we might consider
- **to-gm** is by default equal to 3, it represents the maximum number of good directions related to monster we might consider
- **alpha-m-attack** is by default equal to 0.5 and should be in general between 0 and 1, it represents by default how aggressive we are against monsters
- **alpha-m-direction** is by default equal to 0.7 and should be in general between 0 and 1. It represents the importance that we give to monsters directions based on the score of their heuristic
- **alpha-m-attack** is by default equal to 0.5 and should be in general between 0 and 1, it represents by default how aggressive we are against humans
- **lam-mh-direction** is by default equal to 0.7 and should be in general between 0 and 1, it represents the importance that we give to humans directions based on the score of their heuristic
- **alpha-ins** is by default equal to 0.1, it is related to the score of battles between monsters, the higher it is, the more extreme it is the more aggressive we will be in our attack with the opponent monster, if it is higher that 1, we are insanely aggressive.

3.2.2 Heuristic computation

The process of computing the heuristic is as follow :

1. We start with some preliminary computations: we compute all the possible admissible positions that we can take in the next step. We compute the heuristic for extreme cases : when we disappear completely (heuristic gets - inf) or when there are no opponent monsters left (heuristic gets + inf).
2. We find the admissible groups of humans that we can target. For that, we compute all the distances between the group of monsters and the humans and store them, but if there are more humans than monsters the distance will be equal to the diagonal distance so that this solution won't be choosen or if there exists an adversary that is nearer than us to the group of humans, it is worthless to go towards that direction so the distance will be equal to maximum distance which is the distance of the diagonal. Then, we normalize the distances by the minimum of number of humans and to-gh all multiplied by the diagonal size which is equivalent to the maximum distance in the map. Finally, we rank the groups by distance and we take at most to-gh nearest groups.

3. We decide which direction of group of humans we prefer by assigning a score to it: for each group in the admissible group of humans defined before, we calculate the direction with regard to us, then, we assign to that direction a score which is the product of the number of humans in that group and the nonlinear(1 - distance) which take into account the distance of the group from us, all divided by total number of humans in the whole map. The nonlinear activation function takes a value and a parameter and return the value power that parameter. It is used because we don't value the distances linearly compared to the number of humans or monsters. The non linear activation function has the following expression:

$$\Psi_\alpha(x) = x^\alpha \tag{1}$$

the term that will account for the distance will thus be $\Psi(1 - d_n)$.

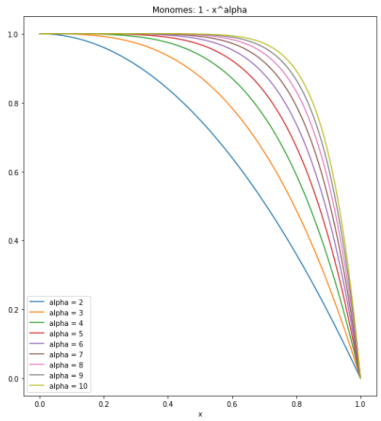

Figure 1: representation of $1 - \Psi_\alpha(x) = 1 - x^\alpha$

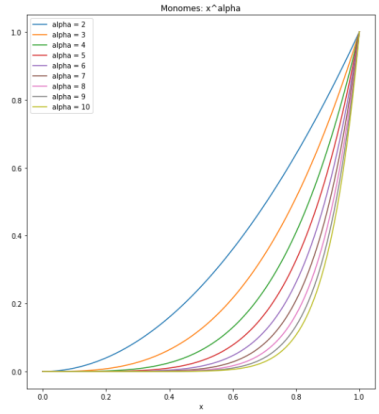

Figure 2: representation of $\Psi_\alpha(x) = x^\alpha$

4. We consequently get a list of heuristic for directions related to humans : heuristic-humans. After, we compute the score of this heuristic by eliminating the forbidden directions and add all the heuristic-human scores by using the score function, then, we get the heuristic-human-score.

5. We compute the expected number of our monster if we attack or not based on the expected gain and the number of humans and monsters in the battle, and we consider it as a score. Then, we standardize this number by the total number of our monsters and the maximum number of humans in the whole map. We consequently get a score for battles with humans : compute-battles-score-humans.

6. We follow the same rational for monster and we add the condition that we don't consider the directions where the number of opponent monsters is higher than 1.5 times our number of monsters. In addition to that, we consider to-gm instead of to-gh. We consider the side case where if there are no humans left and if there are more monsters of the other opponent, we attack in any case even if we will probably loose.

   We consequently get a list of heuristic for directions related to monsters : heuristic-monsters-score and a score for battles with monsters : compute-battles-score-monster.

7. Finally we compute the value of the heuristic : Val-heur = lam-mh-direction * heuristic-human-score + lam-mh-attack * compute-battles-score-humans + alpha-m-direction * heuristic-monsters-score + alpha-m-attack * compute-battles-score-monsters

The expected gain is computed as follow :

a. Expected gain humans: Let's denote $m^{(t)}$ and $h_*^{(t)}$ respectively the number of our monsters and the numbers of the humans of a given group at the instant t ¿ 0. Since we know that probability of winning with the humans P_{E_1,H_1} is defined as:

$$P_{E_1,H_1} = \begin{cases} 1 \, for \, \frac{E_1}{H_1} \geq 1 \\ \frac{\frac{E_1}{H_1}}{2} \, for \, \frac{E_1}{H_1} < 1 \end{cases} \tag{2}$$

then, the expected gain of humans is defined as expectation of the number of monsters at the step $t+1$, which is computed as follows:

$$\begin{aligned} \mathbb{E}[m^{(t+1)}] &= P_{m^{(t)},h_*^{(t)}} \mathbb{E}[m^{(t+1)}|\{Win\}] + (1 - P_{m^{(t)},h_*^{(t)}}) \mathbb{E}[m^{(t+1)}|\{Lose\}] \\ &= P_{m^{(t)},h_*^{(t)}} \left[P_{m^{(t)},h_*^{(t)}} \left(m^{(t)} + h_*^{(t)} \right) \right] + (1 - P)0 \\ &= P^2_{m^{(t)},h_*^{(t)}} \left(m^{(t)} + h_*^{(t)} \right) \end{aligned}$$

In practice, if $\frac{\mathbb{E}[m_1^{(t+1)}]}{m_1^{(t)}} > 1$ the attacking human part of the heuristic will prefer to attack.

b. Expected gain monsters: Let's consider the case of a battle between monsters.Let $m_1^{(t)}$ and $m_2^{(t)}$ respectively the number of attacking monsters and the defending monsters. Following the same rationale as above, we define the expected gain for the attacking monsters as the expectation of the number of attacking monsters at the step $t+1$. Likewise, we define the expected gain for the defensive monsters as the expectation of the number of defensive monsters at the step $t+1$. Again, since we know that probability of winning against the other monsters P_{E_1,E_2} is defined as:

$$P_{E_1,E_2} = \begin{cases} 1 \, for \, \frac{E_1}{E_2} > 1.5 \\ \frac{E_1}{E_2} - \frac{1}{2} \, for \, 1 \leq \frac{E_1}{E_2} \leq 1.5 \\ \frac{\frac{E_1}{E_2}}{2} \, for \, \frac{E_1}{E_2} < 1 \end{cases} \tag{3}$$

the expected number of attacking monsters $m_1^{(t+1)}$ and the expected number of defensive monsters $m_2^{(t+1)}$ will be:

$$\begin{aligned}\mathbb{E}[m_1^{(t+1)}] &= P_{m_1^{(t)},m_2^{(t)}}\mathbb{E}[m_1^{(t+1)}|\{Win\}] + (1 - P_{m_1^{(t)},m_2^{(t)}})\mathbb{E}[m_1^{(t+1)}|\{Lose\}] \\ &= P^2_{m_1^{(t)},m_2^{(t)}} m_1^{(t)} + (1 - P_{m_1^{(t)},m_2^{(t)}})0 \\ &= P^2_{m_1^{(t)},m_2^{(t)}} m_1^{(t)}\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}[m_2^{(t+1)}] &= P_{m_2^{(t)},m_1^{(t)}}\mathbb{E}[m_2^{(t+1)}|\{Win\}] + (1 - P_{m_2^{(t)},m_1^{(t)}})\mathbb{E}[m_2^{(t+1)}|\{Lose\}] \\ &= P^2_{m_2^{(t)},m_1^{(t)}} m_2^{(t)} + (1 - P_{m_2^{(t)},m_1^{(t)}})0 \\ &= P^2_{m_2^{(t)},m_1^{(t)}} m_2^{(t)}\end{aligned}$$

In practice, if $\frac{\mathbb{E}[m_1^{(t+1)}]}{\mathbb{E}[m_2^{(t+1)}]} > \frac{m_1^{(t)}}{m_2^{(t)}}$ the attacking monsters part of the heuristic will prefer to attack.

Case of depth four :

For the depth 4, we keep overall the same logic as for depth two introduced formally. However, new parameters are added : IMPORTANCE-BATTLES which tells how important are the results of the battles with respect the directions. ANTI-COMM which is the price of commutativity, i.e. how much the first move (depth = 2) is more important than the second move (depth = 4), Computing the ratio at the two steps and evaluating the difference. For instance attack humans at the first step (depth = 2) and then move to another cell (depth = 4) must be different than move to another cell (depth = 2) and then attack humans (depth = 4). RATIO-BETWEEN-STATES-2 is ratio number of us in the two steps (between depth = 2 and depth = 4) and RATIO-BETWEEN-STATES-4 (between depth = 0 and depth = 2). It also takes in consideration the two former stages. Those former states are then used in order to calculate the number of monsters at each step. This is used for the purpose of avoiding commutative problems. This process is made efficiently thought additive and multiplicative simple terms. It allowed us to keep a good time complexity and a simplicity in the code.

4 The Algorithm

4.1 Presentation of the algorithm

The python files used are :

- main.py : this script allows to execute the algorithm , and make the connexion with the server that sends the following information:
- alphabeta.py : it implements the alpha-beta algorithm for depth 2
- alphabeta2.py : it implements the alpha-beta algorithm for depth 4
- computesuccessors.py : it enables us to get the successor states of a specific state
- state.py: we use it to define the class state that builds the well defined object state
- heuristic.py : it defines and implements the heuristic used for apha-beta depth 2
- heuristic2.py : it defines and implements the heuristic used for apha-beta depth 4

4.2 Description of the classes

Class ”state” is the main class of the algorithm. This class represents a state, as in a grid configuration. It is a list of lists of 5 numbers. Each chunk of 5 numbers represents a non-empty cell of the grid (position x, position y, nb-humans, nb-vampires, nb-werewolves). height and width are the dimensions of the grid. To initialize this class, we need a list of states, width and height of the grid. The main methods of this state are : new-state :

- new-state :this methods takes a state and a set of modifications, being a list of lists of 5 elements representing cells which have been modified. It updates the state into a new one,taking into account these modifications
- get-nb-humans(self):Method which takes a state and return the number of humans on the board
- get-vampires-list(self) : Method which takes a state and return a list of lists of 3 elements being (position x, position y, nb-vampires)
- get-nb-vampires(self):Method which takes a state and return the number of vampires on the board
- get-werewolves-list(self) : same as get-vampires-list but for werewolves
- get-nb-werewolves(self):same as get-nb-vampires but for werewolves

4.3 Description of the functions

4.3.1 Computation of successors

For the computation of successors, the main functions used are :

- allowed-directions (x, y, w, h): it get a position (x,y) and the width and height of the grid and gives the allowed directions with taking in consideration the borders of the grid
- compute-successors(s, player): it gets the current state of the game and the player that we are (werewolves or vampires) and return successor of state s as [proba, [directions], [successor state]]
- compute-states-after-moves(state, moves, player) : it gets a state, a list of moves [[[direction], [x, y, nb, newx,, newy], ...]], the type of player (werwolves or vampires) and returns a list of [proba, [directions], state]. This helps to calculate the compute-successors fonction.
- from-modifs-to-states(state, modifs): from a list of modifs like this one [[[0.75, [4, 1, 0, 4, 0]], [0.25, [4, 1, 0, 0, 1]]],[1, [7, 7, 0, 4, 0]], [1, [1, 5, 0, 2, 0]], [1, [4, 1, 0, 0, 4]], [1, [2, 3, 4, 0, 0]],[1, [5, 8, 0, 4, 0]], [1, [1, 4, 0, 2, 0]]], it build modifs and returns the new list of states modified
- result(case, nb-player-moved, player):we have a case [x, y nb-h, nb-v, nb-w] and we move nb-player-moved of monsters of categories player in this case: this function return the new case as a tuple: (probability of this case, the case)
- battle-with-monsters(E1,E2): If a battle with monsters happends, it takes as input the number of us E1, and number of opponent monster E2 and returns each case possible as [proba, nb-monsters1, nb-monsters2]
- battle-with-humans(E1,E2) : If a battle with humans happens, it takes as input the number of us E1, and number of opponent humans E2 and returns each case possible as [proba, nb-monsters, nb-humans]

All the functions implemented are implemented twice in a mirror basis , depending on if we play as vampires or werewolves.

4.3.2 Heuristic

For the heuristics, First, we needed to map the directions.We consider for that a mapping invertible function f(x,y)= 3*x + y + 4 and its invert. f(x,y) takes the direction coordinates under the shape (x,y) , x and y can get the values -1, 0 or 1 where : x : -1 (left), 0 (center), 1 (right) y : -1 (down), 0 (middle), 1 (up) Consequently, with f we get the following general mapping : (-1,-1) $\rightarrow 0, (-1,0) \rightarrow 1, (-1,1) \rightarrow 2, (0,-1) \rightarrow 3, (0,0) \rightarrow 4, (0,1) \rightarrow 5, (1,-1) \rightarrow 6, (1,0) \rightarrow 7, (1,1) \rightarrow 8$

The python functions are :

- dir-index-map (d-x,d-y) : the mapping function
- inv-index-map(val): the inverse of the mapping function

Then we calculated the distances.In order to calculate the distances, we start by considering the simple cases then, we combine their results to solve the complex cases. The simple cases are when two points are in the same vertical or horizontal axe, then, their distance is y2-y1 (same vertical axe) or x2-x1(same horizontal axe). Or, when two points are in the same diagonal, then , there distance is x2-x1. Now, if two points are not in the same axe (vertical, horizontal or diagonal), we create a loop where we calculate incrementally the distance by using a fictional moving element from the one location to the other one, and using the distances of the simple cases, the loop stops when the fictional element arrives to the other location

- distance-between-groups-principal-axis(x1,y1,x2,y2): returns the distance between the point 1 (x1,y1) , and 2 (x2,y2) when they are in the same vertical or horizontal axe
- distance-between-groups-diagonals(x1,y1,x2,y2)): returns the distance between the point 1 (x1,y1) , and 2 (x2,y2) when they are in the same diagonal
- distance-between-groups(x1,y1,x2,y2, adj0 = False) : returns the distance between the point 1 (x1,y1) , and 2 (x2,y2) wherever they are. Depending on adj0, we have +-1 to the simple cases.

Following the expected gain strategy introduced before, we calculate the expected gain of a battle between creatures and humans and between creatures and monsters

- expected-gain-humans(E1,H1) : expected gain for a battle between E1 monsters and H1 humans
- expected-gain-monster(E1,E2) : expected gain for a battle between E1 monsters and E2 monsters

Finally, we defined the functions that calculate the heuristics as defined in the former part.

- compute-score(heuristic,index-possible-directions) : it take a list of scores associated to each direction and computes score associated to the possible directions
- compute-score-state-player(our-monsters-list, their-monsters-list, humans-list, our-monsters-nb-groups, their-monsters-nb-groups, nb-groups-humans, nb-our-monsters, width, height, MAX-DIST, MAX-NB-HUMANS, INF, to-gh = 3, to-gm = 3, alpha-m-attack = 0.5 , alpha-m-direction = 0.7, lam-mh-attack = 0.5 , lam-mh-direction = 0.7, alpha-ins = 0.1, print-heuristic = False) : the heuristic function for depth 2 as defined before
- compute-score-state(s-act, s-prec2, s-prec4, player, print-heuristic = False) : the heuristic function for depth 2 as defined before

4.3.3 Alpha-beta

- max-value(state, alpha, beta, player, depth, depth-max, best-direction=None, best-state=None): it gets as an input a state, the current alpha and beta values, the depth we are in and the max depth, the best direction and depth that are None by default, and returns the max value list by applying the alpha-beta algorithm. For a depth 4, it takes the current state in addition to the three former ones.
- min-value(state, alpha, beta, player, depth, depth-max, best-direction=None, best-state=None): it takes the same input as max-value and returns the minimum value list by applying the alpha-beta algorithm .For a depth 4, it takes the current state in addition to the three former ones.
- compute-best-direction (state, alpha, beta, player, depth, depth-max) :it takes the same input as max-value, prepossesses, gets the max value then returns the best moves to follow
- from-direction-to-move(alpha-beta-result, intermediary-state, player): it gets a list resulting from max-value function, the state and the type of player we are, then returns the list of moves.

5 Conclusion

Difficulties and points of improvement

5.1 Successor states

- Difficulty in visualizing what we were doing when we want to obtain successors of successors
- Difficulty of implementation: combinatorial aspect, indices, management of the edges of the grid (on the edges, the number of movements allowed is less)
- Management of all possible cases: calculating the new state after a movement. Depending on the type of battle, depending on the case, depending on the situation of the other species, depending on the number of species groups involved... Not easy to take everything into account. In particular, a big difficulty in this function was to take into account all the situations: the squares that become empty after a movement must be removed from the state, the squares that did not yet exist in the state must be created, following a battle, there can be disappearance of a square without adding another...
- Difficulty in managing random battles (how to consider successors: do we consider two different states of different probabilities depending on the outcome of the random battle, or do we consider a single state with the expected number of vampires, werewolves and humans on it? In order to be able to use the rules of the game without having to invent new ones when looking at the successors of a state resulting from a random battle, it is absolutely necessary that no species cohabit on a square: we have therefore chosen strategy 1 (several states of different probabilities).
- We have made the choice not to allow our groups of our species to separate. The door is open because our implementation is consistent with this, but given the difficulty of building alpha-beta and heuristics, we have chosen to focus first on a good implementation of these two parts. There is also a very heavy combinatorial aspect when we allow our groups to separate which can prevent a good evaluation of our implementation at first.

5.2 Alpha-Beta

- At the beginning, we were working on the implementation of a Tree data structure with nodes etc. We had written classes. Obviously, in front of the implementation difficulties, it was not the right solution and we went back to an alpha-beta as written in the course.
- Difficulty: we do not want to know what is the best successor state to our current state but rather what is the best direction our species must take to achieve this best state. Because of the battles and the possibility of having several groups, it is difficult to find the direction that our species has followed between two successive states. It was therefore necessary to keep in mind in the alpha-beta the best direction to follow, in addition to the best condition.
- A probabilistic dimension has been included in the alpha-beta: since we have chosen to consider the successor states of random battles as probabilistic states, they are associated with a certain proba. It was also necessary to keep the proba of the state in memory in the alpha-beta. The upward heuristic was then multiplied by the proba of the state in order to take into account this probabilistic dimension.

5.3 Heuristic

- Difficulty in defining and selecting all the parameters to be taken into account
- Difficulty in removing the oscillations of the heuristics that made our species only oscillate between two squares rather than eat the enemies.
- At the beginning, evaluation of the heuristic directly in the alpha beta: not the right solution because the alpha beta "covered" too much the movements that the heuristics required us to do. So, setting up a reflex agent to evaluate the heuristic before combining alpha-beta and heuristics.