Game theory project: strategies for the game Troll&Castles

Guilhem MARION, Robert KALNA

Mai 2018

1 Project

This project shows a way to compute efficiently strategies for the game *Troll and Castle* using game theory and linear programming. The idea is to formalize each of the rounds with game theory and find the mixed strategies that maximize the worth case (prudent strategy) using linear programming.

This way of doing actually minimize the loss but don't try to maximize the gain. That's why we have chosen to implement the elimination of dominated strategies. We claim that this way of doing is capable of maximize the gain in the case of a game against a *good player*. Be a *good player* means to be able to play the strategies that are the best for you in terms of mathematical expectation.

Therefore, this way of deciding is the best in term of expectation, in other terms, the strategies computed are the best in convergence, because of the Law of Large numbers. Due to the definition of convergence, we cannot ensure for a finite number of games that our strategy will be better than another one.

To try to win faster in some cases we can try to guess the strategy of the other player in term of conditional probabilities (i.e. the probability of playing a strategy depending on the current state). Then, we can find a strategy that maximizes our gain. The idea is that allows to win faster and surely with a greater gain in the case that we have a good guess of the strategy of the other player.

For that, we can use Machine Learning (Markov Chains of order n, Neural Networks, ...) to learn the distribution while we are playing the prudent strategy, when the guessed distribution is converging very well we use it to choose what to play. This part is not implemented and can also be very risky because of all the unknown parameters of the system (the player can change strategy, he can use Machine Learning to find a better strategy against the one we found, convergence time can be very long on certain strategies, ...).

You can find here simulations we have made against other strategies: you will find that we won all the other strategies! But chance is an important parameter and can change a little bit the issue of some games.

2 Strategies

2.1 Random number of stones

```
def strategy_random(game, previous_parties):
        number_of_stones_of_enemy = min(game.stockGauche, game.stockDroite + 1)
        return int(np.random.choice(range(1, number_of_stones_of_enemy + 1)))
```

2.2 Always throw 2 stones

```
def strategy_always_throw_two(game, previous_parties):
        number_of_stones = game.stockGauche
        return min(2, number_of_stones)
```

2.3 Gaussian with location of 2 and variance of 0.5

```
def strategy_gaussian(game, previous_parties):
        stones_to_throw = np.random.normal(2, 0.5)
        if stones_to_throw > game.stockGauche:
```

```
                stones_to_throw = min(np.random.normal(game.stockGauche//2, 3), game.stockGauche)
        return int(max(stones_to_throw, 1))
```

2.4 Strategy leading to Nash equilibrium

Distributions have been calculated before and stored in pickles: `distributions` is an object loaded from a pickle. There is one for games with 7 fields and another with 15 fields. The tables of the utilities have also been calculated and stored in pickles, see `field7/utilities.pkl` and `field15/utilities.pkl`.

```
def strategy_of_nash(game, previous_parties):
        troll_position = int(game.positionTroll - (game.nombreCases - 1) // 2)
        stones_left = game.stockGauche
        stones_right = game.stockDroite
        if (stones_left, stones_right, troll_position) in distributions:
                ((distribution, distribution_ind), g) = distributions[
                                        stones_left, stones_right, troll_position]
        else:
                ((distribution, distribution_ind), g) = db.calculate_what_to_play(
e                                stones_left, stones_right, troll_position)
        distribution = np.array(distribution)
        distribution /= distribution.sum()
        X = np.random.choice(distribution_ind, 1, p=distribution)
        return int(X[0])
```

2.5 Eager version of the strategy leading to Nash equilibrium

```
def strategy_nash_eager(game, previous_parties):
        troll_position = int(game.positionTroll - (game.nombreCases - 1) // 2)
        stones_left = game.stockGauche
        stones_right = game.stockDroite
        if (stones_left, stones_right, troll_position) in distributions:
                ((distribution, distribution_ind), g) = distributions[
                                        stones_left, stones_right, troll_position]
        else:
                ((distribution, distribution_ind), g) = db.calculate_what_to_play(
                                        stones_left, stones_right, troll_position)
        return int(np.array(distribution).argmax() + 1)
```

3 Number of fields: 7, stones: 15

3.1 Strategy of nash VS random number of stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 965
Victoires du joueur de droite : 21
Matchs nuls : 14

Victoire du joueur de gauche !

3.2 Strategy of nash VS eager version of strategy of nash

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 304
Victoires du joueur de droite : 271
Matchs nuls : 425

Victoire du joueur de gauche !

3.3 Strategy of nash VS gaussian with location of 2 and variance of 0.5

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 716
Victoires du joueur de droite : 212
Matchs nuls : 72

Victoire du joueur de gauche !

3.4 Strategy of nash VS always throw two stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 372
Victoires du joueur de droite : 343
Matchs nuls : 285

Victoire du joueur de gauche !

3.5 Nash equilibrium

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 337
Victoires du joueur de droite : 329
Matchs nuls : 334

Victoire du joueur de gauche !

4 Number of fields: 7, stones: 30

4.1 Strategy of nash VS random number of stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 952
Victoires du joueur de droite : 46
Matchs nuls : 2

Victoire du joueur de gauche !

4.2 Strategy of nash VS eager version of strategy of nash

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 475
Victoires du joueur de droite : 431
Matchs nuls : 94

Victoire du joueur de gauche !

4.3 Strategy of nash VS gaussian with location of 2 and variance of 0.5

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 851
Victoires du joueur de droite : 149
Matchs nuls : 0

Victoire du joueur de gauche !

4.4 Strategy of nash VS always throw two stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 659
Victoires du joueur de droite : 341
Matchs nuls : 0

Victoire du joueur de gauche !

4.5 Nash equilibrium

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 487
Victoires du joueur de droite : 411
Matchs nuls : 102

Victoire du joueur de gauche !

5 Number of fields: 15, stones: 30

5.1 Strategy of nash VS random number of stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 1000
Victoires du joueur de droite : 0
Matchs nuls : 0

Victoire du joueur de gauche !

5.2 Strategy of nash VS eager version of strategy of nash

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 3
Victoires du joueur de droite : 0
Matchs nuls : 997

Victoire du joueur de gauche !

5.3 Strategy of nash VS gaussian with location of 2 and variance of 0.5

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 497
Victoires du joueur de droite : 216
Matchs nuls : 287

Victoire du joueur de gauche !

5.4 Strategy of nash VS always throw two stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 1
Victoires du joueur de droite : 0
Matchs nuls : 999

Victoire du joueur de gauche !

5.5 Nash equilibrium

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 2
Victoires du joueur de droite : 1
Matchs nuls : 997

Victoire du joueur de gauche !

6 Number of fields: 15, stones: 50

6.1 Strategy of nash VS random number of stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 1000
Victoires du joueur de droite : 0
Matchs nuls : 0

Victoire du joueur de gauche !

6.2 Strategy of nash VS eager version of strategy of nash

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 311
Victoires du joueur de droite : 270
Matchs nuls : 419

Victoire du joueur de gauche !

6.3 Strategy of nash VS gaussian with location of 2 and variance of 0.5

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 726
Victoires du joueur de droite : 262
Matchs nuls : 12

Victoire du joueur de gauche !

6.4 Strategy of nash VS always throw two stones

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 210
Victoires du joueur de droite : 53
Matchs nuls : 737

Victoire du joueur de gauche !

6.5 Nash equilibrium

----- Resultats de la simulation -----

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 364
Victoires du joueur de droite : 349
Matchs nuls : 287

Victoire du joueur de gauche !