Game theory project: strategies for the game Troll&Castles

Guilhem MARION, Robert KALNA

Mai 2018

1 Strategies

This project shows a way to compute efficient strategies for the Troll and Castle game using game theory and linear programming. The idea is to formlize each of the round with game theory and find the strategies that maximize the worth case using leanar programming.

The way of doing actually minimize the loss but don't try to maximize the gain. That's why we choose to implement the elimination of dominated strategies. We claim that this way of doing is capable of maximise the gain in the case of a game against a *good player*. Be a *good player* means be able to play the strategies that is the best for you in terme 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 then an other one. That's why some strategies can win the one we compute on 1000 repetitions.

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 of the current state). Then, we can compute of distribution in the strategies that maximise the gain and the gain in the worth case. The idea is that allow to win faster and surely with a greather gain in the case that we have a good guess of the strategy of the other player and that he wont change it during the game. 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, and when the guessed distribution is convenging very well use it to choose what to play knowing it. 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, can also use Machine Learning to find a better strategy then us, convergence time can be very long on certain strategies, ...).

You can find here simulations we made with other strategies.

1.1 Random number of stones

```python
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)))
```

1.2 Always throw 2 stones

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

1.3 Gaussian with location of 2 and variance of 0.5

```python
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))
```

1.4 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])
```

1.5 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)
```

2 Number of fields: 7, stones: 15

2.1 Strategy of nash VS random number of stones

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 951
Victoires du joueur de droite : 42
Matchs nuls : 7

Victoire du joueur de gauche !

2.2 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 : 275
Victoires du joueur de droite : 283
Matchs nuls : 442

Victoire du joueur de droite !

2.3 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 : 675
Victoires du joueur de droite : 275
Matchs nuls : 50

Victoire du joueur de gauche !

2.4 Strategy of nash VS always throw two stones

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 342
Victoires du joueur de droite : 378
Matchs nuls : 280

Victoire du joueur de droite !

2.5 Nash equilibrium

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 309
Victoires du joueur de droite : 324
Matchs nuls : 367

Victoire du joueur de droite !

3 Number of fields: 7, stones: 30

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 : 947

Victoires du joueur de droite : 50
Matchs nuls : 3

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 : 438
Victoires du joueur de droite : 457
Matchs nuls : 105

Victoire du joueur de droite !

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 : 732
Victoires du joueur de droite : 268
Matchs nuls : 0

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 : 617
Victoires du joueur de droite : 383
Matchs nuls : 0

Victoire du joueur de gauche !

3.5 Nash equilibrium

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 451
Victoires du joueur de droite : 491
Matchs nuls : 58

Victoire du joueur de droite !

4 Number of fields: 15, 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 : 1000
Victoires du joueur de droite : 0
Matchs nuls : 0

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

Victoire du joueur de droite !

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 : 476
Victoires du joueur de droite : 214
Matchs nuls : 310

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

Victoire du joueur de gauche !

4.5 Nash equilibrium

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 4
Victoires du joueur de droite : 2
Matchs nuls : 994

Victoire du joueur de gauche !

5 Number of fields: 15, stones: 50

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 : 303
Victoires du joueur de droite : 243
Matchs nuls : 454

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 : 734
Victoires du joueur de droite : 254
Matchs nuls : 12

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 : 177
Victoires du joueur de droite : 65
Matchs nuls : 758

Victoire du joueur de gauche !

5.5 Nash equilibrium

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

Matchs prevus : 1000
Matchs joues : 1000

Victoires du joueur de gauche : 330
Victoires du joueur de droite : 355
Matchs nuls : 315

Victoire du joueur de droite !