Deep Catan

Brahim Driss and Tristan Cazenave

LAMSADE, Université Paris-Dauphine, PSL, CNRS, Paris, France
Tristan.Cazenave@dauphine.psl.eu

Abstract. Catan is a popular multiplayer board game that involves multiple game-play notions: stochastic elements related to the dice rolls as well as to the the initial placement of resources on the map and the drawing of development cards, strategic notions for the placement of the cities and the roads which call upon topological and shape recognition notions and notions of expectation of gains linked to the probabilities of the rolls of the dice. In this paper, we develop a policy for this game using a convolutional neural network. The used deep reinforcement learning algorithm is Expert Iteration [2] which has already given excellent results for Alpha Zero and its descendants.

1 Introduction

Monte Carlo Tree Search (MCTS) [7, 11] has been used in two-player complete information games. Modern board games such as Catan have more complex rules and deal with incomplete information due to dice rolls or drawing cards. MCTS has already been applied to Catan with success [15]. In this paper we address the use of deep neural network in combination with MCTS to play Catan. The combination of Deep Reinforcement Learning with MCTS gave strong computer players for Go, Chess and Shogi with Alpha Zero [14] and was further applied to many games with the Polygames framework [6]. The underlying Deep Reinforcement Learning for these systems is Expert Iteration [2]. In this paper we advocate that the combination of deep neural networks trained from zero knowledge in combination with MCTS can outperform MCTS alone.

The paper is organized as follows. The second section recalls related work. The third section presents Deep Reinforcement Learning of Catan. The fourth section gives experimental results.

2 Background and Related Work

2.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a general search algorithm that was initially designed for the game of Go [7]. The most popular MCTS algorithm is Upper Confidence bounds applied to Trees (UCT) [11]. UCT is the standard MCTS algorithm. It uses the mean of the previous random playouts to guide the beginning of the current playouts. There is a balance between exploration and exploitation when choosing the next move to try at the beginning of a playout. Exploitation tends to choose the move with the best mean, while exploration tends to try alternative and less explored moves to see if they can become better. The principle of UCT is optimism in face of uncertainty. It chooses the action with the UCB formula:

$$argmax_a \left\{ Q(s,a) + C\sqrt{\frac{log(N(s))}{N(s,a)}} \right\} \tag{1}$$

where $N(s,a)$ is the number of simulations of the node, $N(s)$ the number of simulations of the parent node (state s before taking action a), $Q(s,a)$ the winrate of the action a in the state s (number of wins/number of simulations) and C the UCB bandit exploration coefficient.
The All Moves As First heuristic (AMAF) [3] is a heuristic that was used in Gobble, the first Monte Carlo Go program [5]. It consists in updating the statistics of the moves of a position with the result of a playout, taking into account all the moves that were played in the playout and not only the first one.
In Catan, standard MCTS with 10,000 simulations could still be beaten easily, [15] used a heuristic action selection procedure inside the MCTS and virtual wins.

In order to improve the level of play, instead of randomly sampling moves, probabilities can be associated to moves according to their type, building a city or settlement having higher chances when available to be selected than building the road for example and skipping the turn being the lowest, since building is always rewarding for the player and skipping when building is available is generally a bad idea.

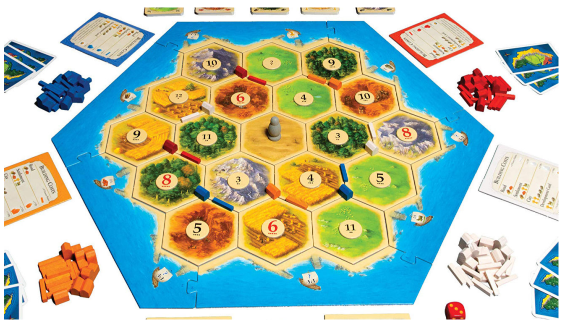

Fig. 1: A Catan board

2.2 Rules of Catan

In Catan, players compete to colonize an island represented by a board (Figure 1) of hexagonal tiles. There are 5 resource types – Brick, Lumber, Ore, Grain, and Wool – which can be spent to make various actions. The first player to reach 10 Victory Points (VP) or more is considered the winner. VP can be acquired by various means: placing settlements (1VP) or cities (2VP) on the board, having the longest road or largest army (2VP), or special development cards (1VP). The island of Catan is represented as a board of 19 land hexagonal tiles called hexes, randomly placed when setting up the game. Tiles can either represent a desert, or produce one of the 5 resources, in which case they will be assigned a number between 2 and 12. We will call the edge of a hex a path, and its corner an intersection.

At the beginning of the game, each player places 2 settlements, each with an adjacent road. Settlements must be placed on intersections and can not be next to one another. During each turn, a player can take a sequence of actions if they have the required resources for it:

- Build a Road : 1*Brick + 1*Lumber
- Build a Settlement : 1*Brick + 1*Lumber + 1*Grain + 1*Wool
- Build a City : 3*Ore + 1*Grain
- Buy a Development Card : 1*Ore + 1*Grain + 1*Wool
- Trade resources with the bank.

For trading the default ratio is four of the same resource for any one resource, but having a settlement or city on a harbor can reduce the rate to 3:1 or 2:1. There is no simple winning strategy. Basically, a stable and varied production of resources is beneficial to obtain VP. Thus, players should prioritize placing their settlements in intersections surrounded by balanced resources and high production chance (with numbers around 7), near promising unexploited areas or on interesting harbors. Buying development cards is a bit of a lucky dip as the player is buying blindly from a stack of cards but they are all beneficial to him. Development cards have different bonuses such as blocking one hex from producing resources and stealing a card from one of its neighbors, adding a victory point, building 2 free roads for example.

2.3 Related Work

Previous works on Catan used various methods. The earliest agent used Model Trees trained through self-play [13], multi-agent systems were also able to obtain strategic game play [4]. Szita, Chaslot, and Spronck [15] used Monte Carlo Tree Search in a perfect-information variation of the game. Other works explored other aspects of the game. Afantenos [1] focuses on strategic conversation concerning bargaining negotiation in the game of Catan. Guhe [10] focuses on persuasions using empirical data from game simulations from a game theory point of view. Other papers also used Deep Reinforcement Learning, two of them focused only on a subset of actions that is trading, using Deep Q-Learning [8] and Deep Q-Learning with LSTM [16]. A third paper [9] also used Deep Reinforcement Learning with an agent using a variation of Advantage Actor Critic in a 2 player version of the game.
In this paper, we use Monte Carlo Tree Search combined with deep neural networks within the original rules, i.e, imperfect information and 4-player game, without domain knowledge, refusing and never initiating trades with other players, but continuing to trade with the bank. The used methods and objectives are different from previous works, since there is no Advantage Actor Critic [12] and the game is not the 2-player version [9].We think that the 2-player version is very different from the 4-player one. Having 3 players playing before the next turn is not the same as having only one. There

will be a higher chance for instance to be blocked by enemies and lose good spots for settlements, there will also be less place to expand because of building constraints. We are also playing the entire game not only focusing on a single part of it, on the contrary of [8] and [16]. Using Monte Carlo Tree Search [15] did not give good results after 10,000 simulations, this is why we propose a Local Value Estimation network that improves on Monte Carlo Tree Search, learning AMAF statistics during self-play. We also show that Monte Carlo Tree Search can be further improved using deep reinforcement learning with Expert Iteration.

3 Deep Reinforcement Learning of Catan

3.1 Supervised Learning of the value network

We train a value network using games played by MCTS. The value network takes a random state of the game as an input and predicts the MCTS winrates of the root nood for the 4 players given by the average of the rollouts from that state. The output of the network is a softmax over the 4 outputs giving the winning probabilities for the 4 players.

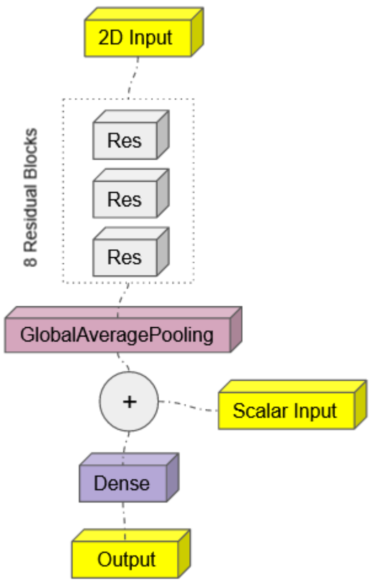

Fig. 2: Value network architecture

Since Catan has multiple information unrelated to the board (e.g visible victory points, development cards or resources left), the network used two inputs: A 2-dimensional input for the board and a scalar input for game information. The 2 dimensional input is transformed using residual blocks to process the board. A regular board of Catan contains 19 hexes, 72 paths and 54 intersections. We employ a similar architecture to the Alpha Zero network. We pass in the board position as a 23×13 image and use convolutions to construct a representation of the position.

Board (2D channels)		29
Roads		1(x4)
Settlements		1(x4)
Cities		1(x4)
Ports		2
Resources types		1(x5)
Resources odds		1(x5)
Robber odds		1(x5)
Vector input		
Game phase		12
Visible VP		11(x4)
Resources cards		11(x4)
Developpment cards		11(x4)
Ressources left		21(x5)
Largest Army		1(x4)
Longest Road		1(x4)
Buildings left		
	Roads left	16(x4)
	Settlements left	6(x4)
	Cities left	5(x4)

Table 1: Neural network 2D input channels

The Catan board is different from boards in games such as Chess and Go. In these games all cells are similar. Catan cells are hexagons and the board topology of Catan is unlike that of Hex or Havannah for instance. The hexes, path and intersections have different roles and features. We split features of different types in 29 channels to prevent the convolution from processing them in the same way. The description of the different channels is shown in Table 1.

We also use the brick coordinate [9]. We use a 5×3 kernel which makes the neighbors comparable to the actual neighbors on the hexagonal board. The mapping is explained in Figure 3.

The kernel used in the convolutions of the residual blocks is the 5x3 brick coordinate kernel. The optimizer is Adam (Learning rate = 0.001), ReLU activation functions in the hidden layers with Dropout (0.3 and 0.5 rate), Softmax in the output layer. The loss is the Mean Absolute Error.

3.2 Local Value Estimation

A second neural network with dense layers and ReLU activations, was trained on AMAF statistics obtained during rollouts. The purpose of this neural network is to eval-

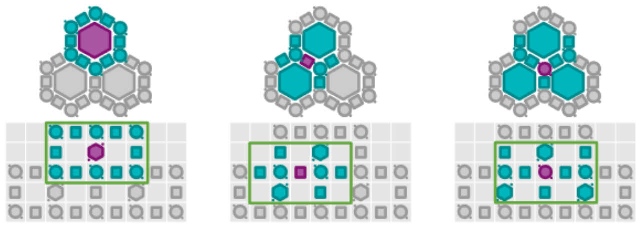

Fig. 3: 5x3 kernel on brick coordinate

uate the player moves to reduce the breadth of the search in the tree, without removing the simulations in the playout phase, by modifying the UCB bandit value, adding an evaluation term:

$$argmax_a \left\{ Q(s,a) + C\sqrt{\frac{log(N(s))}{N(s,a)}} + C_2 \times eval \right\} \tag{2}$$

where $eval$ is this neural network prediction of the value of the move. Instead of focusing on the board as a whole state, the network evaluates the possible moves locally, giving more data to train on that is less complex (n pairs of moves and scores instead of a single state and its value). It is updated after each training iteration with data from the self-play games. The inputs of the network are the moves and their local corresponding features as shown in Figure 4 and the output is the prediction of the AMAF score of MCTS playouts. For buildings, the neighbor hexes of the construction are the features. This network will be used in the UCTNet experiment and be compared to UCT without the network evaluation, provided with the same budget of rollouts.

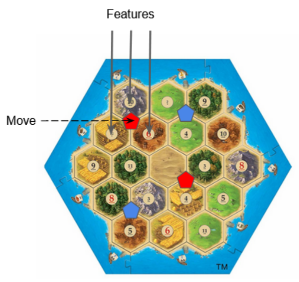

Fig. 4: Input example for the Local Value network

3.3 Expert Iteration of the value network

Compared to imitation learning techniques, Expert Iteration (ExIt) is enriched by an expert improvement step. Improving the expert player and then solving the imitation learning problem allows us to exploit the fast convergence properties of imitation learning, even in contexts where no strong player was originally known, such as when learning from scratch.

At each iteration i, the algorithm proceeds as follows: we create a set S_i of game states by playing the $\hat{\pi}_{i-1}$ learner. In each of these states, we use our expert to compute an imitation learning target at s (e.g., the expert's $\pi^*_{i-1}(a|s)$ action); the state-target pairs (e.g., $(s, \pi^*_{i-1}(a|s))$) form our dataset D_i. We train a new apprentice $\hat{\pi}_i$ on D_i (learning by imitation). Then, we use our new apprentice to update our expert $\pi^*_i = \pi^*(a|s; \hat{\pi}_i)$ (expert improvement).

4 Experimental Results

4.1 Importance of the budget

The budget allocated to the different MCTS is important. An experiment was performed to verify the impact of the number of rollouts on the MCTS performance. Four matches of 200 games were performed, opposing an improved MCTS (number of rollouts multiplied by 2) against 3 normal MCTS. The results of this experiment are shown in Table 2:

Match	Games won	Winrate
100 vs 50	70	35%
200 vs 100	56	28%
400 vs 200	63	31.5%
800 vs 400	59	29.5%

Table 2: Matches between MCTS using twice as many rollouts as its 3 opponents.

We can see that the MCTS with the most rollouts always has a positive winrate (higher than 25%). A higher number of rollouts improves the level of the MCTS.

4.2 Training performance

First iteration The neural network is trained on the scores of 800 games played in self-play (4 different players using the same method in order to find the best move) and evaluated on 200 games that do not exist in the training data at each ExIt iteration. To avoid having similar states, all games are mixed, and the network is trained on mini-batches of 64 states. At the end of the iteration, the network will have learned about 50,000 game states with their Monte Carlo scores.

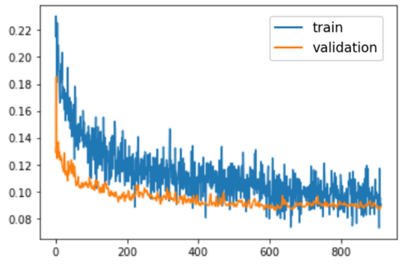

Fig. 5: Evolution of the network loss during training at the 1st iteration

Figure 5 shows that at the beginning of the training the network has an average loss of 0.2 in the predictions which eventually stabilizes at 0.1 at the end of the training over the 1,000 games played. The network is then able to make Monte Carlo score predictions for the 4 players with an average error of 0.1 without simulating the game.

Second iteration The second iteration is the step where the network trained in the first iteration is used in an MCTS and replaces the rollouts by a single neural network evaluation.

The new scores obtained are the labels of a new neural network. The new neural network is then trained on 1,000 games of self-play by MCTS using the first network.

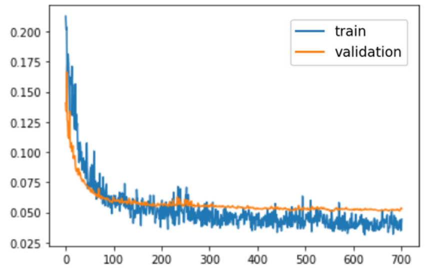

Fig. 6: Evolution of the network loss during training at the 2nd iteration

We notice as shown in the Figure 6 that the network starts with the same error of 0.2 and stabilizes at a lower error than the first iteration (0.05 in the test compared to 0.1 in the first iteration).

We also notice that the training of the new network is less noisy, with smaller variance of the error during the training.

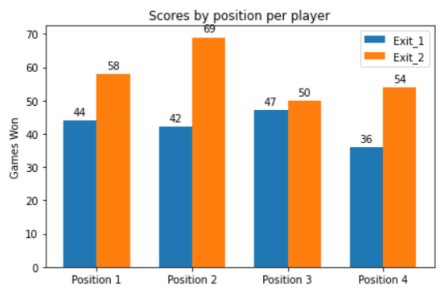

Fig. 7: Results of the games played between the two models

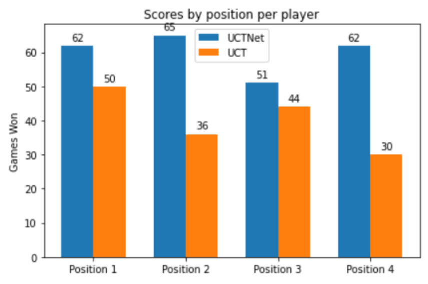

Fig. 8: Results of the games played between UCT and UCTNet

4.3 Evaluation of neural networks

Value network 400 games opposing the two value networks (ExIt_1 is the first iteration and ExIt_2 the second) in 2 vs 2 were played, with the same budget of rollouts given to the two networks, the positions (1,2,3,4) of the players are drawn randomly at the beginning of the game. The network of the second iteration won 231 games out of 400 played (58% winrate) against the network of the first iteration. Results of the games are shown in Figure 7.

Local Value Estimation Network 400 games opposing UCT (Classic MCTS algorithm) and UCTNet (UCT using the Local Value Estimation), with the same budget of rollouts given to the two UCT, the positions (1,2,3,4) of the players are drawn randomly at the beginning of the game. UCTNet won 240 games out of 400 played (60% winrate) against UCT. The results are shown in Figure 8.

5 Conclusion and Future Work

The first experiment shows that Expert Iteration can improve the level of play in a game involving chance. Since the Monte Carlo evaluations of the network were less noisy,

the training in the second iterations had less variance. The second experiment evaluates moves instead of entire boards and therefore was easier to train. Using such a network, UCTNet was able to defeat UCT in 60% of games.
For future work, we plan to improve the current environment. Firstly, we are using Python, the code can be further improved to give faster simulations in order to do more iterations of our reinforcement algorithm. Almost 50,000 samples were needed for the value network at each iteration. Secondly, trading between players is a social and interesting part of the game. Adding trading is the next step, which would require some changes in the game loop and an update to the MCTS structure, since trading involves multiple players in the same turn.

Acknowledgment

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

References

1. Afantenos, S., Asher, N., Benamara, F., Cadilhac, A., Dégremont, C., Denis, P., Guhe, M., Keizer, S., Lascarides, A., Lemon, O., et al.: Developing a corpus of strategic conversation in the settlers of catan. In: SeineDial 2012-The 16th Workshop On The Semantics and Pragmatics Of Dialogue (2012)
2. Anthony, T., Tian, Z., Barber, D.: Thinking fast and slow with deep learning and tree search. In: Advances in Neural Information Processing Systems. pp. 5360–5370 (2017)
3. Bouzy, B., Helmstetter, B.: Monte-Carlo Go developments. In: ACG. IFIP, vol. 263, pp. 159–174. Kluwer (2003)
4. Branca, L., Johansson, S.J.: Using multi-agent system technologies in settlers of catan bots. In: Agent-based Systems for Human Learning and Entertainment (ABSHLE) (2007)
5. Brügmann, B.: Monte Carlo Go. Tech. rep. (1993)
6. Cazenave, T., Chen, Y.C., Chen, G.W., Chen, S.Y., Chiu, X.D., Dehos, J., Elsa, M., Gong, Q., Hu, H., Khalidov, V., Cheng-Ling, L., Lin, H.I., Lin, Y.J., Martinet, X., Mella, V., Rapin, J., Roziere, B., Synnaeve, G., Teytaud, F., Teytaud, O., Ye, S.C., Ye, Y.J., Yen, S.J., Zagoruyko, S.: Polygames: Improved zero learning. ICGA Journal **42**(4), 244–256 (December 2020)
7. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers. pp. 72–83 (2006)
8. Cuayáhuitl, H., Keizer, S., Lemon, O.: Strategic dialogue management via deep reinforcement learning. vol. abs/1511.08099. Springer (2015)
9. Gendre, Q., Kaneko, T.: Playing catan with cross-dimensional neural network. In: Neural Information Processing. (ICONIP 2020). pp. 580–592. Springer (2020)
10. Guhe, M., Lascarides, A.: The effectiveness of persuasion in the settlers of catan. In: 2014 IEEE Conference on Computational Intelligence and Games. pp. 1–8. IEEE (2014)
11. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings. pp. 282–293 (2006)

12. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 1928–1937. PMLR, New York, New York, USA (20–22 Jun 2016), https://proceedings.mlr.press/v48/mniha16.html
13. Pfeiffer, M.: Reinforcement learning of strategies for settlers of catan. In: Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education. Citeseer (2004)
14. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D.: Mastering chess and shogi by self-play with a general reinforcement learning algorithm. CoRR **abs/1712.01815** (2017)
15. Szita, I., Chaslot, G., Spronck, P.: Monte-carlo tree search in settlers of catan. In: Advances in Computer Games. pp. 21–32. Springer (2009)
16. Xenou, K., Chalkiadakis, G., Afantenos, S.: Deep reinforcement learning in strategic board game environments. In: Slavkovik, M. (ed.) Multi-Agent Systems. pp. 233—-248. Springer (2019)