

Monte Carlo Tree Search for Carcassonne

Bachelor's Thesis

Max Jappert <max.jappert@unibas.ch>

Department of Mathematics and Computer Science
University of Basel

27.06.2022

Outline

1. Introduction
2. Monte Carlo Tree Search
3. Implementation and Evaluation
4. Conclusion

Why Monte Carlo Tree Search?

- › Monte Carlo Tree Search (MCTS) has been successfully applied to:
 - › Hex
 - › Lines of Action
 - › Settlers of Catan
 - › Go
- › Hayden (2009) and Amenityro et al. (2020) have suggested that MCTS produces good results on Carcassonne.
- › They don't consider different variants.

Why Monte Carlo Tree Search?

- › Monte Carlo Tree Search (MCTS) has been successfully applied to:
 - › Hex
 - › Lines of Action
 - › Settlers of Catan
 - › Go
- › Hayden (2009) and Amenityro et al. (2020) have suggested that MCTS produces good results on Carcassonne.
- › They don't consider different variants.

Why Monte Carlo Tree Search?

- › Monte Carlo Tree Search (MCTS) has been successfully applied to:
 - › Hex
 - › Lines of Action
 - › Settlers of Catan
 - › Go
- › Hayden (2009) and Amenityro et al. (2020) have suggested that MCTS produces good results on Carcassonne.
- › They don't consider different variants.

Research Objectives

- › Evaluating different variants of Monte Carlo Tree Search in regard to their performance on Carcassonne.
- › Evaluating if the most powerful variant is capable of beating a human player.

Carcassonne

- › Carcassonne is a tile-based board game for between two and five players.
- › The board is iteratively built by placing tiles over the course of 72 rounds.
- › Points are made by placing *meeples* strategically.
- › Large state space with at least $5 \cdot 10^{40}$ reachable positions and a game tree with around 10^{192} terminal nodes (Heyden, 2009).

Source: <https://www.dadsgamingaddiction.com/carcassonne/>
(23.06.2022)

Carcassonne

- › Carcassonne is a tile-based board game for between two and five players.
- › The board is iteratively built by placing tiles over the course of 72 rounds.
- › Points are made by placing *meeples* strategically.
- › Large state space with at least $5 \cdot 10^{40}$ reachable positions and a game tree with around 10^{192} terminal nodes (Heyden, 2009).

Source: <https://www.dadsgamingaddiction.com/carcassonne/>
(23.06.2022)

Carcassonne

Carcassonne is a tile-based board game for between two and five players.

The board is iteratively built by placing tiles over the course of 72 rounds.

Points are made by placing meeples strategically.

Large state space with at least $5 \cdot 10^{40}$ reachable positions and a game tree with around 10^{192} terminal nodes (Heyden, 2009).

Source: <https://www.dadsgamingaddiction.com/carcassonne/>
(23.06.2022)

Carcassonne

Carcassonne is a tile-based board game for between two and five players.

The board is iteratively built by placing tiles over the course of 72 rounds.

Points are made by placing meeples strategically.

Large state space with at least $5 \cdot 10^{40}$ reachable positions and a game tree with around 10^{192} terminal nodes (Heyden, 2009).

Source: <https://www.dadsgamingaddiction.com/carcassonne/>
(23.06.2022)

Monte Carlo Tree Search

MCTS is a method for finding optimal decisions in a given domain.

It "combines the precision of tree search with the generality of random sampling" (Browne et al., 2012, p. 1).

This is achieved by taking random samples in the state space and building a search tree according to the results.

Monte Carlo Tree Search

MCTS is a method for finding optimal decisions in a given domain.

It "combines the precision of tree search with the generality of random sampling" (Browne et al., 2012, p. 1).

This is achieved by taking random samples in the state space and building a search tree according to the results.

Monte Carlo Tree Search

MCTS is a method for finding optimal decisions in a given domain.

It "combines the precision of tree search with the generality of random sampling" (Browne et al., 2012, p. 1).

This is achieved by taking random samples in the state space and building a search tree according to the results.

Training

During training, a game tree is iteratively built.

Each node has a visit count N and an expected score Q , which get updated during training.

Each training iteration consists of four steps.

Training

During training, a game tree is iteratively built.

Each node has a visit count N and an expected score Q , which get updated during training.

Each training iteration consists of four steps.

Training

During training, a game tree is iteratively built.

Each node has a visit count N and an expected score Q , which get updated during training.

Each training iteration consists of four steps.

Step 1: Selection

Traverse the game tree according to the ~~tree~~ policy, which maps each node to one of its children.

Do this until an expandable node is reached.

Image Source: James et al. (2017), p. 2

Step 2: Expansion

Add at least one child node and visit that child.

Image Source: James et al. (2017), p. 2

Step 3: Simulation

Simulate the game until the end with moves decided by the default policy

Thereby sample a score.

The simulation can be replaced by a function with domain-specific knowledge.

Image Source: James et al. (2017), p. 2

Step 4: Backpropagation

The sampled score R is propagated back up the chosen path. For each node:

$$\begin{aligned} N & \quad N + 1 \\ Q & \quad Q + \frac{R - Q}{N} \end{aligned}$$

Image Source: James et al. (2017), p. 2

Practical Steps

1. Implementing Carcassonne.
2. Implementing the MCTS framework.
3. Testing and evaluating different MCTS configurations.

Implementation of Carcassonne

Implemented in Java.

Allows for MCTS variants and humans to play Carcassonne.

The repository is publicly accessible under https://github.com/maxjappert/mcts_carcassonne

Implementation: Game Tree

Chance nodes: Randomly drawing a tile

Placement nodes: Placing a tile

Meeple nodes: Placing a meeple

Implementation: Ensemble MCTS

Alternative way of modelling randomness.

k trees are built, whereby each assumes a fixed deck permutation.

After training all trees "vote" on which move to pick.

Evaluation: Single Game Tree vs. Ensemble MCTS

Certain Ensemble MCTS configurations lead to performance increase.

Implementation: Degree of Exploration

All "useful" tree policies are a function of the children's Q -value, because the game tree should be expanded in profitable directions.

All "useful" tree policies must balance this exploitation with a degree of exploration.

Most tree policies have an exploration parameter, which determines the degree of exploration.

Implementation: Degree of Exploration

All "useful" tree policies are a function of the children's Q -value, because the game tree should be expanded in profitable directions.

All "useful" tree policies must balance this exploitation with a degree of exploration.

Most tree policies have an exploration parameter, which determines the degree of exploration.

Implementation: Degree of Exploration

All "useful" tree policies are a function of the children's Q -value, because the game tree should be expanded in profitable directions.

All "useful" tree policies must balance this exploitation with a degree of exploration.

Most tree policies have an exploration parameter, which determines the degree of exploration.

Evaluation: Degree of Exploration

Severe performance drop when exploration seizes.

Evaluation: Tree Policies

Measured \ Opponent	UCT	UCT-Tuned	Boltzmann	"-Greedy	Dec. "-G.	Heur. MCTS	Heuristic	Random
UCT	{	38.2%	63.9%	44.4%	72.2%	83.3%	81.1%	100.0%
UCT-Tuned	61.8%	{	70.3%	51.4%	80.6%	86.1%	77.8%	100.0%
Boltzmann	36.1%	29.7%	{	27.8%	81.1%	63.9%	62.9%	100.0%
"-Greedy	55.6%	48.6%	72.2%	{	86.5%	83.8%	78.4%	100.0%
Decaying"-Greedy	27.8%	19.4%	18.9%	13.5%	{	31.4%	51.4%	89.2%
Heuristic MCTS	16.7%	13.9%	36.1%	16.2%	68.6%	{	59.5%	100.0%
Heuristic	18.9%	22.2%	37.1%	21.6%	48.6%	40.5%	{	100.0%
Random	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	8.6%	{

UCT-Tuned performed best.

Implementation: Simulation Step

Usually consists of randomly selecting moves.

Adding domain-specific knowledge can potentially improve performance.

We tested two alternatives:

- Heuristic default policy

- Direct heuristic evaluation

Evaluation: Heuristic Default Policy

A heuristic-guided playout increases performance slightly, but also increases the runtime by a factor of 30.

Evaluation: Direct-Heuristic Evaluation

Decreases runtime per training iteration by a factor of 30 compared to using random sampling.

Performed slightly worse than a random playout when considering a similar runtime.

Evaluation: Playing against MCTS

We played six games against the strongest implementation.
It won five of those games with an average score of 95.3 to 88.3.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $\alpha = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.

- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $c_t = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.

- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $\alpha = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.

- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $\alpha = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.

- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $c_t = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.

- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

MCTS produces good results on the domain of Carcassonne.

We found a variant which can outperform an average human player. This variant has the following properties:

- UCT-Tuned as a tree policy.

- Decaying exploration parameter $c_t = 512/t$ for t training iterations.

- Simulation with a random default policy.

- Ensemble MCTS with four trees, 750 training iterations each.

Interesting for further research:

- Considering probability distribution over the deck in the game tree.


- Optimising the heuristic functions.

- Training a neural network to approximate the score given a state.

- Bounding the simulation.

Conclusion

- › MCTS produces good results on the domain of Carcassonne.
- › We found a variant which can outperform an average human player. This variant has the following properties:
 - › UCT-Tuned as a tree policy.
 - › Decaying exploration parameter of $c = 512/t$ for t training iterations.
 - › Simulation with a random default policy.
 - › Ensemble MCTS with four trees, 750 training iterations each.
- › Interesting for further research:
 - › Considering probability distribution over the deck in the game tree.
 - › Optimising the heuristic functions.
 - › Training a neural network to approximate the score given a state.
 - › Bounding the simulation.



Thank you
for your attention.