

**Computer Games Development**

**Project Report**

**Year IV**

[Zhenze Zhao]

[C00198551]

[03/05/2020]

# **Faculty of Computing and Networking Science**

# **Open-Book and Remote Assessment Cover Page**

# 

# **Student Name:Zhenze Zhao**

# 

# **Student Number: C00198551**

# 

# **Lecturer Name: Lei Shi**

# 

# **Module:Final Year Project**

# 

# **Stage/Year:4th**

# 

# **Date:03/05/20**

# 

# **Declaration**

# 

# This examination/assessment will be submitted using Turnitin as the online submission tool. By submitting my examination/assessment to Turnitin, I am declaring that this examination/assessment is my own work. I understand that I may be required to orally defend any of my answers, to the lecturer, at a given time after the examination/assessment has been completed, as outlined in the student regulations.

# 

# **Project Abstract**

This project is about testing the Monte Carlo Tree Search working with a game that requires huge calculation.

The current algorithm is used to simulate possibilities and find the best solution from those possibilities. What I want to do for this project is develop a complex game, then test the AI efficiency in this game. Then trying to find any algorithm that can help the Monte Carlo Tree Search to finish the game and increase the efficiency and speed.

In the end, I will have another AI algorithm to compare with Monte Carlo Tree Search. Showing a result that shows how long, and accuracy for AI moves. Then I can get the advantage of the Monte Carlo Tree Search and when I should use this AI algorithm.

My solution is to develop two AI with different algorithms. Then show how fast the AI can get the answer with the algorithm and the AI needs to finish a go game.

# **Project Introduction and Research Question**

The traditional Go game is on a 19 \* 19 board, that means the first move of the game has 391 possibilities. And the rules of the game give the ability to take off the opponent’s pieces. So in the common Go game, the reasonable possibilities is 2.081681994 \* 10^170. This value for any normal AI algorithm is impossible. But AlphaGo did it, and it can find the answer really fast.

The purpose of this project is to find how AlphaGo works with the Monte Carlo Tree Search function. And why AlphaGo needs other algorithms to find the answer. After Project finish, The Monte Carlo Tree Search will compare with the AlphaBeta in a small board to test the efficiency and accuracy of the AI. This study will see how the Monte Carlo Tree Search does the simulation. And answer the question below.

Answering the questions:

* How can the Monte Carlo Tree Search handle a game with huge simulations?
* How can the Monte Carlo Tree Search work in a limited performance environment?
* Can AI just work with Monte Carlo Tree Search, or does the AI need other algorithm’s help to work?

# **Background**

After I watch the go match with AlphaGo, I try to find out what algorithms AlphaGo used. In the 26 algorithms that AlphaGo used, the most important part is Policy network and Value network, Monte Carlo Tree Search and Reinforcement Learn. For the Monte Carlo Tree Search, the algorithm’s can handle the complex situation and find the best result in limited calculation. The Precision and efficiency of the Monte Carlo Tree Search can be used in the field like Medical treatment, Statistical Physics, Prospecting and so on. For this project I want to develop an AI with Monte Carlo Tree Search for Unity Go game just like AlphaGo . I would then research and look into existing Unity and C# documents, forums and videos with helping the AI Monte Carlo Tree Search run in a computer with limited performance. And create an AI pattern for Unity Go game.

# **Literature Review**

# ***“Monte-Carlo tree search in AlphaGo.***

1. *Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus a bonus u(P) that depends on a stored prior probability P for the edge.*
2. *The leaf node may be expanded; the new node is processed once by the policy network pσ and the output probabilities are stored as prior probabilities P for each action.*
3. *At the end of a simulation, the leaf node is evaluated in two ways: using the value network vθ; and by running a rollout to the end of the game with the fast rollout policy pπ,then computing the winner with function r.*
4. *Action-values Q are updated to track the mean value of all evaluations r(·)and vθ(·)in the subtree below that action.*

*For the single best move. However, the value function vθ(s)≈vpρ(s) derived from the stronger RL policy network performed better in AlphaGo than a value function vθ(s)≈vpρ(s) derived from the SL policy network.*

*Evaluating policy and value networks requires several orders of magnitude more computation than traditional search heuristics. To efﬁciently combine MCTS with deep neural networks, AlphaGo uses an asynchronous multi-threaded search that executes simulations on CPUs, and computes policy and value networks in parallel on GPUs. The ﬁnal version of AlphaGo used 40 search threads, 48 CPUs, and 8 GPUs. We also implemented a distributed version of AlphaGo that exploited multiple machines, 40 search threads, 1202 CPUs and 176 GPUs. The Methods section provides full details of asynchronous and distributed MCTS.****”***

[***Source***](https://www.researchgate.net/publication/292074166_Mastering_the_game_of_Go_with_deep_neural_networks_and_tree_search)

This shows the Monte Carlo tree search in AlphaGo, how it works and why it gets the answer in 5sec. The policy network helped the AI get the win rate for all nodes on the board and the tree search chose the high rate node to simulate the answer.

***2.2 Stochastic Bandit Problems and UCB***

***A bandit problem withKarms (actions) is defined by the sequence of random pay offs Xit,i=1,...,K,t≥1, where each is the index of a gambling machine2The functionbestMoveis trivial, and is omitted due to the lack of space.***

***Bandit Based Monte-Carlo Planning2851:***

***functionMonteCarloPlanning(state)2:repeat3:search(state,0)4:untilTimeout5:returnbestAction(state,0)6:function search(state,depth)7:ifTerminal(state)then return08:ifLeaf(state, d)then returnEvaluate(state)9:action:= selectAction(state,depth)10: (nextstate, reward) := simulateAction(state,action)11:q:=reward+γsearch(nextstate,depth+1)12: UpdateValue(state, action, q, depth)13:return Fig. 1.The pseudocode of a generic Monte-Carlo planning algorithm(the “arm” of a bandit). Successive plays of machine yield the payoffsXi1,Xi2,.... For simplicity, we shall assume that X it lies in the interval [0,1]. An Allocation policy is a mapping that selects the next arm to be played based on the sequence of past selections and payoffs obtained. The expected regret of an allocation policy afternplays is defined byRn=maxiE[∑nt=1Xit]−E[∑Kj=1∑Tj(n)t=1Xj,t],whereIt∈{1,...,K}is the index of the arm selected at time t by policyA,andwhere Ti(t)=∑ts=1I(Is=i)is the number timesarmiwas played up to timet(including t). Thus, the regret is the loss caused by the policy not always playing the best machine. For a large class of payoff distributions, there is no policy whose regret would grow slower thanO(lnn)[9]. For such payoff distributions, a policy is said to resolve the exploration-exploitation tradeoff if its regret growth rate is within a constant factor of the best possible regret rate.Algorithm UCB1, whose finite-time regret is studied in details by [1] is a simple, yet attractive algorithm that succeeds in resolving the exploration-exploitation tradeoff. It keeps track the average rewardsXi,Ti(t−1)for all the arms and chooses the arm with the best upper confidence bound:It=argmax∈{1,...,K}{Xi,Ti(t−1)+ct−1,Ti(t−1)},(1)where ct,sis a bias sequence chosen to be ct,s=√2lnts.(2)The bias sequence is such that if X it were independently and identically distrib-uted then the inequalitiesP(Xis≥μi+ct,s)≤t−4,(3)P(Xis≤μi−ct,s)≤t−4(4)were satisfied. This follows from Hoeffding’s inequality. In our case, UCB1 is used in the internal nodes to select the actions to be sampled next. Since for***

***286L. Kocsis and C. Szepesv ́ariany given node, the sampling probability of the actions at nodes below the node(in the tree) is changing, the payoff sequences experienced will drift in time.Hence, in UCT, the above expression for the bias terms ct, s needs to replaced by a term that takes into account this drift of payoffs. One of our main results will show despite this drift, bias terms of the form ct,s=2Cp√lnts with appropriate constants Cp can still be constructed for the payoff sequences experienced at anyof the internal nodes such that the above tail inequalities are still satisfied.***

[***Source***](https://link.springer.com/content/pdf/10.1007%2F11871842_29.pdf)

To make the answer more precise, the UCB help the AI choose the node from most times visited node or less times visited node.

# **Study**

# **Minimax**: the mini max is a decision rule used in ai. The maxmin value of a player is the highest value that the player can be sure to get without knowing the actions of the other players. The lowest value the other players can force the player to receive when AI knows the player’s action. The Minimax is Minimize your opponent's advantage and maximize my advantage.

Pseudocode example:

Function minimax(node, depth)

If node is a terminal node or depth = 0

Return the heuristic value of node

If the adversary is to play at node

Let a =+ ∞

Foreach child of node

A = min(a, minimax(child, depth -1))

Else {we are to play at node}

Let a= -∞

Foreach child of node

A = max (a, minimax(child, depth-1))

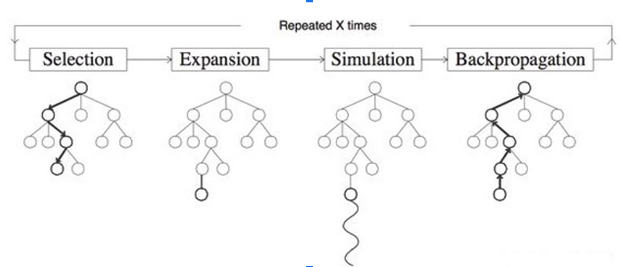
Return a;

The depth in the pseudocode is a limit for how many times the code runs. The code will stop running when the depth reduces to 0 or find the final answer leaf which means win the game or lose the game.

**The monte carlo tree search(MCTS):**The method is using randomness for deterministic problems difficult or impossible to solve using other approaches.

There are 4 steps for this method:

1. Selection : find a node as start. The best choice is the leaf node which no simulation has yet initiated.
2. Expansion: create a new child node for the leaf node we selected at selection step.
3. Simulation: let the node from Expansion stop as start, and simulate the game until the end of the game. Then get the score from that node.
4. Backpropagation: return the score from the node which selected from Expansion step to the parent nodes. And upload the node’s quality value and visit times.

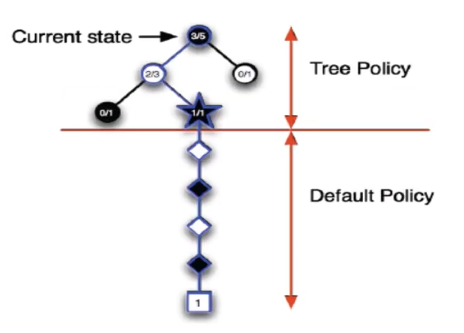
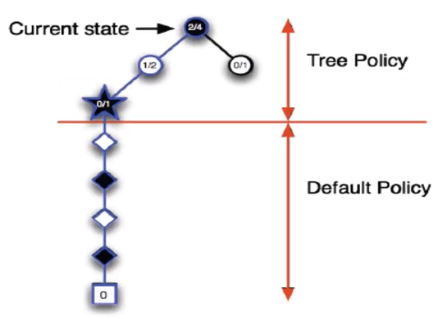
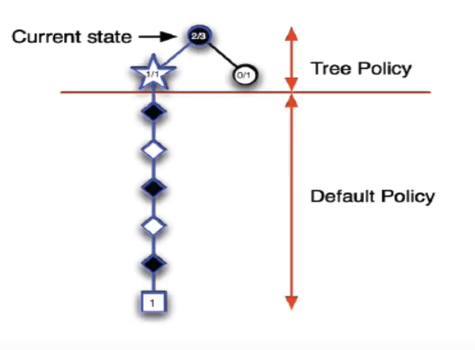
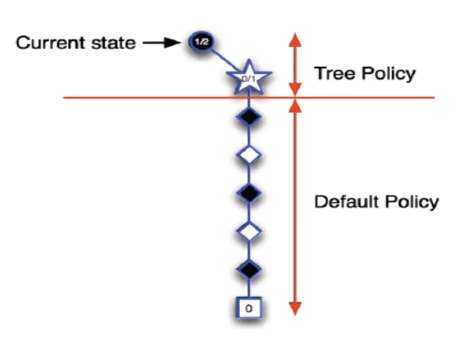


The method will get the most of leaf nodes’ UCB value, then the next simulation will be based on these values.

The Monte Carlo Tree Search can be summarized as 2 policies: Tree Policy and Default Policy.

Tree Policy: Select or create a leaf node from the search tree.

Default Policy: Simulate the game to the end by starting from an unknown node.

****

Pseudocode example:

Function UCTSearch(s0)

Create root node v0 with current board state s0

While within computational budget do

V1 = TreePolicy(v0)

△ = DefaultPolicy(s(v1))

BackUp(v1, △)

Return a(BestChild(v0,0))

Function TreePolicy(v)

While v is nonterminal do

If v not fully expanded

Return Expand(v)

Else

V = BestChild (v, Cp)

Return v

Function DefaultPolicy(s)

While s is non-terminal do

Choose a node from A(s) uniformly at random

S = f(s,a)

Return reward for state s

Function Expand(v)

Choose a node from untried actions from A(s(v))

Add a new child v’ to v

With s(v’) = f(s(v),a) and a(v’) = a

Return v’

Function BackUp(v,△)

While v is not null do

Visited time ++

Score = Score + final win/lose

V = parent of v

Function BestChild(v,c)

Return UCB value

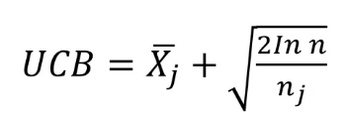
**The Upper Confidence Bound(UCB) algorithm:** The algorithm is based on the principle of optimism in the face of uncertainty, which is to choose your actions as if the environment is as nice as is plausible possible.

And for this algorithm we need to know the Exploitation and Exploration.

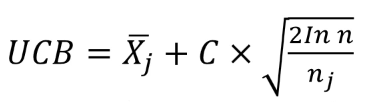
Exploitation: Make sure the past decisions have the most advantageous paths/results.

Exploration: Looks for more advantageous paths/results from future decisions.

The UCB algorithm is balanced with exploration and exploration to find the best way to get the results. And the UCB value follows the function below.



Xj is the average score for the current node, nj is the number of times visited current node, n is the overall number of all simulations.



To make the AI behaviour based on the exploitation or the exploration. We can use a C value to control the AI behaviour.

**The Policy Network and Value Network:**

The policy network: the user input we make as A(action), because this input the game’s output we called S(state). After we statistics all input and we will have a state list for the action. And the network can offer an output list based on the input.

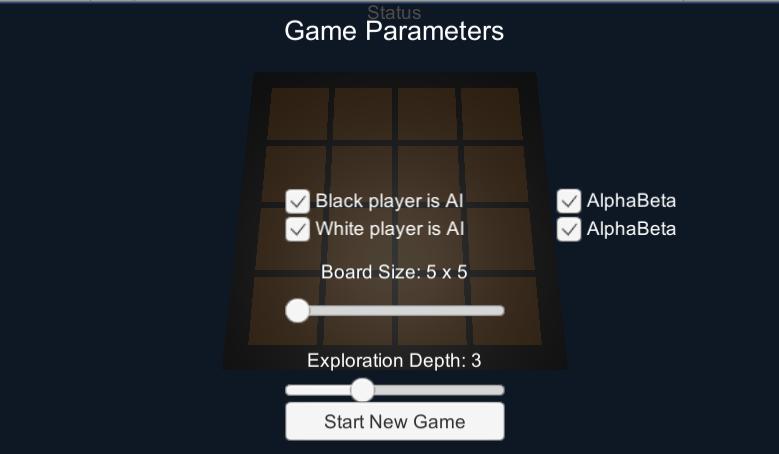


The Value Network: the network will calculate the expected rewards based on the current state. The bigger rewards value in the network means this state is more valuable.



# **Project Description**

This project is basically a Go game including AI. The main ai algorithm is the Monte Carlo Tree Search. To make the AI working more efficiently and precisely, I try to add the other algorithm to help the AI make choices and get answers. In the end of the project, I developed a simple AI with AlphaBeta to learn the differences between 2 AI and see which one is more efficient and precise.



For this project, it allows the user to choose the game board size with the Board Size Slider and who controls the player. The AI function can also be selected by the user. It shows the AI work in different situations, and the difference between those two AI.

In this project, I have learned a lot about coding and implementing the AI algorithm. I work on a desktop computer and it will limit the performance for AI simulation. The limited performance causes the running time of the algorithm. And this time taken shows the effectiveness of each AI function. The final win rate shows the accuracy of the function.Technically, after the game difficulty increased. The AlphaBeta will take a really long time for one answer, but the accuracy won’t change. But the Monte Carlo Tree Search is different, the effectiveness advantage will show up and the accuracy of the answer is based on the support function.

Personally, the Monte Carlo Tree Search algorithm needs other functions to increase the accuracy. If the AI only runs with the Monte Carlo Tree Search, the accuracy of the answer will be low, because the simulation is totally random. If the search tree is based on the low value node, the answer will be wrong. So the Monte Carlo Tree Search needs the algorithm to scrap unnecessary nodes, it saves time and increases the accuracy of answers.

# **Project Milestones**

Milestones of AI project:

Game build with game loop, update and render in Unity-（15th November 2019）

* Set up of the game
* Game modeling

Game board created-(27th November 2019)

* Create a background board

Game piece-(10th December 2019)

* Create piece prefab for game
* Hold the position and color data

Implement Go game rule-(22nd January 2020)

* The Player can place piece with mouse click and the turn change after input
* Create a invisible board in front t of the background board
* The system remove the pieces if these get surrounded by opponent’s pieces
* The system checks the available move, if it is 0 end the game.
* Generate the available node list for players.

Create node for Monte Carlo Tree Search-(28th February 2020)

* The node keep the necessary states
* Create new child node for next move

Insert Monte Carlo Tree Search-(19th March 2020)

* Give the AI player Monte Carlo Tree Search function
* Use the tree policy to choose start node for simulation
* Use the default policy to simulate the game to end at the copy of the current board
* Find the most valuable node in the search tree as next move

Implement menu for game-(25th March 2020)

* The menu set the board size and player stats

Insert UCB algorithm for tree search-(6th April 2020)

* Give the AI UCB value to control the AI behavior and answer selection.

Insert AlpaBeta AI-(20th April 2020)

* Add selection for chose AI style
* Create AlpaBeta AI function for player
* The AlpaBeta algorithm can search all possibilities for the game and find the best answer.

# Finalise Game-(29th April 2020)

* Make final changes and set the value for final presentation and demo.

# **Results and Discussion**

|  |  |  |  |
| --- | --- | --- | --- |
|  | AlphaBeta(depth:3) | AlphaBeta(depth:7) | MCTS(100 nodes) |
| 5\*5 | 0.92 sec/ per move | More than 10 min | 8.75sec / per move |
| 7\*7 | 5.34 sec/ per move | More than 10 min | 34.21sec / per move |
| 9\*9 | 41.03 sec / per move | More than 10 min | 1:14 min / per move |
| 17\*17 | More than 10 min | More than 10 min | 6:13 min / per move |

Here we can see a table of results showing the effectiveness of different AI algorithms from the different board size(simulation possibilities, The complexity of the game ).

When the AlphaBeta AI depth is 3 and work with the smaller board(5\*5), the AI can get the answer really fast. With the same board, the Monte Carlo Tree Search algorithm can get the answer but it takes longer(is about 10 times) . Check with the AlphaBeta with depth 7, unfortunately my pc may not be able to get the answer in 10 minutes, I tried 3 times but I didn't get an answer once. So we can see when the number of simulation samples are small, the time AI algorithm needed is short. But the answer with small simulation samples is inaccurate.

The bigger board with 9\*9, the AlphaBeta with small depth needs more time ro get the next move. The AlphaBeta with large depth still can’t get an answer in time. Back to the Monte Carlo Tree Search AI, this algorithm time taken still longer than AlphaBeta but it only 1.55 times.

The biggest board with 17\*17, the AlphaBeta is not able to find an answer with at least 3 depth. The Monte Carlo Tree Search spand 6 minutes and 13 seconds, but still get the answer from the search tree.

# **\***I realised the node selection of the Monte Carlo Tree Search is too random, if the max calculation nodes value is small the deviation of the answer will be large. So I am trying to add the Upper confidence Bound function, policy network and value network to the CMTS AI.

# **Project Review and Conclusions**

Let’s start with what went right:

1. The Monte Carlo Tree Search algorithm work with Go game

As we can see from the results, the Monte Carlo Tree Search is able to handle the basic Go game with 19 \* 19 board.

The reason Monte Carlo Tree Search can solve the puzzle in time is the algorithm will not simulate all possibilities at once. The default value setting decides how many expansions nodes will have. For explem, if I set the expansion to 100 when the search tree expands to 100 leaf nodes, the algorithm simulation will stop then find the best node from these 100 expanded nodes. If I set the search time to 2 minutes, the expand node function will stop simulation at 2 minutes and find the best node from the search tree.

1. Insert the Upper Confidence Bound to MCTS

Before I insert the UCB to MCTS, the best node will be the most visited node. The UCB value can control the MCTS select node based on the current states. When the score of current states is less than the opponent, and the most visited node can’t help the AI to win the game. The control value(C) of the UCB algorithm will control the AI that chose the least visited node(but expected rewards value is high) as the next move.

1. Compare with the AlphaBeta function

Back to the result, the AlphaBeta sacrifices the deviation to get the efficiency advantage. And why the Monte Carlo Tree Search has high efficiency. Because the tree search won’t calculate all situations, the other algorithm will help the search tree scrap the low value and unnecessary nodes. That keeps the AI working with high efficiency. Back to see the research question 3, Can AI just work with Monte Carlo Tree Search? My answer must be “NO”. The Monte Carlo Tree Search is just a random simulation function. The deviation of random simulation won't be smaller than calculating all possibilities. Only the other algorithm can help the Monte Carlo Tree Search AI work with high efficiency and low deviation.

# What went wrong:

1. The value network and policy network are not insert to MCTS function

To make the value network and policy network help the Monte Carlo Tree Search choose a position for simulation. It needs a large amount of data of existing Go game history to build the Action-State list for policy networks. And when I realize the AI need those algorithm is monthly deadlines.

# **References**

|  |  |  |
| --- | --- | --- |
| **Referenced Publication** | **Citation** | **Reference** |
| Video Course | (Author Year)  Examples  <https://www.youtube.com/watch?v=niIaKaWIRX0&t=1156s> | The Monte Carlo Simulation |
| Web-site | <https://www.researchgate.net/publication/292074166_Mastering_the_game_of_Go_with_deep_neural_networks_and_tree_search>  Author(s) Marc Lanctot. Christopher Maddison | Author(s) Marc Lanctot. Christopher Maddison  Mastering the game of Go with Deep Neural Networks and Tree Search, Nature. |
| Web-site | <http://www.incompleteideas.net/609%20dropbox/other%20readings%20and%20resources/MCTS-survey.pdf>  Cameron Browne  VOL. 4, NO. 1, MARCH 2012 | Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in games |
| Book | <https://link.springer.com/chapter/10.1007/11871842_29>  Part of the [Lecture Notes in Computer Science](https://link.springer.com/bookseries/558) book series (LNCS, volume 4212) | Authors:Levente Kocsis,Csaba Szepesvári Bandit Based Monte-Carlo Planning |
| Web-Site | <https://arxiv.org/abs/1611.00625>  (Submitted on 26 Aug 2019, last revised 10 Oct 2019 (this version, v4)) | Author(s) -[Marc Lanctot](https://arxiv.org/search/cs?searchtype=author&query=Lanctot%2C+M), [Edward Lockhart](https://arxiv.org/search/cs?searchtype=author&query=Lockhart%2C+E), [Jean-Baptiste Lespiau](https://arxiv.org/search/cs?searchtype=author&query=Lespiau%2C+J), [Vinicius Zambaldi](https://arxiv.org/search/cs?searchtype=author&query=Zambaldi%2C+V), [Satyaki Upadhyay](https://arxiv.org/search/cs?searchtype=author&query=Upadhyay%2C+S), [Julien Pérolat](https://arxiv.org/search/cs?searchtype=author&query=P%C3%A9rolat%2C+J), [Sriram Srinivasan](https://arxiv.org/search/cs?searchtype=author&query=Srinivasan%2C+S), [Finbarr Timbers](https://arxiv.org/search/cs?searchtype=author&query=Timbers%2C+F), [Karl Tuyls](https://arxiv.org/search/cs?searchtype=author&query=Tuyls%2C+K), [Shayegan Omidshafiei](https://arxiv.org/search/cs?searchtype=author&query=Omidshafiei%2C+S), [Daniel Hennes](https://arxiv.org/search/cs?searchtype=author&query=Hennes%2C+D), [Dustin Morrill](https://arxiv.org/search/cs?searchtype=author&query=Morrill%2C+D), [Paul Muller](https://arxiv.org/search/cs?searchtype=author&query=Muller%2C+P), [Timo Ewalds](https://arxiv.org/search/cs?searchtype=author&query=Ewalds%2C+T), [Ryan Faulkner](https://arxiv.org/search/cs?searchtype=author&query=Faulkner%2C+R), [János Kramár](https://arxiv.org/search/cs?searchtype=author&query=Kram%C3%A1r%2C+J), [Bart De Vylder](https://arxiv.org/search/cs?searchtype=author&query=De+Vylder%2C+B), [Brennan Saeta](https://arxiv.org/search/cs?searchtype=author&query=Saeta%2C+B), [James Bradbury](https://arxiv.org/search/cs?searchtype=author&query=Bradbury%2C+J), [David Ding](https://arxiv.org/search/cs?searchtype=author&query=Ding%2C+D), [Sebastian Borgeaud](https://arxiv.org/search/cs?searchtype=author&query=Borgeaud%2C+S), [Matthew Lai](https://arxiv.org/search/cs?searchtype=author&query=Lai%2C+M), [Julian Schrittwieser](https://arxiv.org/search/cs?searchtype=author&query=Schrittwieser%2C+J), [Thomas Anthony](https://arxiv.org/search/cs?searchtype=author&query=Anthony%2C+T), [Edward Hughes](https://arxiv.org/search/cs?searchtype=author&query=Hughes%2C+E), [Ivo Danihelka](https://arxiv.org/search/cs?searchtype=author&query=Danihelka%2C+I), [Jonah Ryan-Davis](https://arxiv.org/search/cs?searchtype=author&query=Ryan-Davis%2C+J) TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games |