

Computer Games Development CW208

GDD and Project Report

Year III

|  |
| --- |
| [ZhenzeZhao]  [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.

Contents

[Project Abstract](#_heading=h.30j0zll) 3

[Project Introduction](#_heading=h.1fob9te) 3

[Background](#_heading=h.3znysh7) 3

[Project Description 4](#_heading=h.2et92p0)

[Game Overview 5](#_heading=h.tyjcwt)

Features5

[Project Milestones](#_heading=h.3vac5uf) 6

[Project Review and Conclusions](#_heading=h.2afmg28) 8

[References](#_heading=h.pkwqa1) 9

# 

# **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**

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.

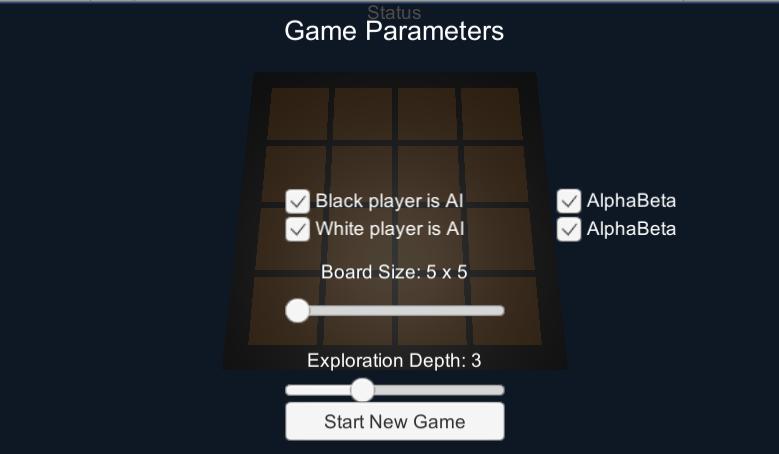
The purpose of this project is to find how AlphaGo works with the Monte Carlo Tree Search function. And why AlphaGo needs other algorithms to find the answer. After Project finish, The Monte Carlo Tree Search will compare with the AlphaBeta in a small board to test the efficiency and accuracy of the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

# **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.

# **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.

# **Game Overview**

This game is basically a Go board game. The game includes 2 different AI and the user can play with the AI player. Also the user can choose to play with humans. At the start of the game, the menu will give the user some options for board size, player state and AI style. The biggest board is 17 \* 17 and the smallest board is 5 \* 5. The player can be controlled by AI or user. There are 2 selections for algorithms for AI, one is the Monte Carlo Tree Search. This algorithm is used to simulate the game with a large number of possibilities，and it simulates with limited search trees and finds the best node from the tree . Another one is the AlphaBeta function. This algorithm will calculate all possibilities in the game, then give the score for each move. After the function records all scores for the nodes, the system will compare the score and find the best move.

The game manager gets the input setting from the player and starts a new game. The generator creates a board and 2 players for the game. After Black player places a piece, the game manager shifts the turn to White player. When a player moves, the board will generate a new point list for the available move. If the player is AI, the board sends a copy of the current state to the AI script. Then the AI starts simulating the game with invisible board, when the simulator ends the AI function delect the copy board and gets a new copy of the current board for simulation.

The board will check the piece on the board, if any pieces have been surrounded by the opponent ‘s, the pieces will be removed from the board. The game manager will check the available list every turn. If the number of the available list reduces to 0, the game ends. And the board counts the pieces on the board as score for each player, the more pieces on the board will win. Then the user can press the ESC key to reopen the menu for a new setting of the game.

When I am coding this project, I know the Monte Carlo Tree Search can not work alone so I try to add the other algorithm to supper the tree search function.The Upper Confidence Bound can reduce the deviation of current simulation answer.

# **Feature Set**

## Feature: Board Generation

This Feature allows the user to set the size of the board and apply settings to the game board. And users can select the AI state, choose the AI run with Monte Carlo Tree Search function or run with AlphaBeta function.

## Feature: Game Board

This feature holds the current game like piece position on the board. And return the available moves to the player. After the player moves, the function will check if this move allows and if this move will take off the opponent pieces.

## Feature: Game Manager

This feature will handle the turn order of each player, shift turn when the player places a piece on the board. The manager handles the player state and checks if the game ends.

## Feature: Player

This feature will place the piece on the board. If the user controls the player, the user’s mouse input will place the piece. If the AI controls the player, the AI will simulate the game with the current algorithm and find the most valuable point as the next player moves.

## Feature: Start Menu

This feature handles the game option from user input, and sand the option data to the generation scripts. After the user setting states, there is a button can start game with current setting.

## Feature: AlphaBeta Search

This feature will use the copy of the current board to simulate the game. The simulation will make sure the player’s next move has the most advantage, and let the opponent only get a few scores. Then give all node’s scores and compare all of the nodes to find the best choice for the next move.

## Feature: Monte Carlo Tree Search

This feature will use the copy of the current board to simulate the game. The function will create an empty node as start. Then start to simulate a random node as a child node, do this step until to the end of the game. After simulation ends, add the value to score and times visited to every node in this search tree. Then simulate ends, give all nodes UCB value and compare UCB value of all nodes to find the next move.

## Feature: Search Node

This feature handles the Monte Carlo Tree Search node state. This function holds the position, score and times visited. This function creates a child node for parents node and holds the copy of the simulation board.

# **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-(25th November 2019)

* Create a background board
* Create a invisible board in front t of the background board

Game piece-(10th December 2019)

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

Implement Go game rule-(20th January 2020)

* The Player can place piece with mouse click and the turn change after input
* 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 states

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 best answer.

# Finalise Game-(29th April 2020)

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

# **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

# 

# **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 |