Department of Computer Science

BSCCS Final Year Project Report
2020-2021

20CS152

Deep Reinforcement Learning in the game of "7 Wonders"

(Volume 3 of 3)

Student Name : **DAWIEANG Phudis**

Student No. : **55411086**

Programme Code : **BSCEGU4**

For Official Use Only

Supervisor : **Prof CHAN, Antoni Bert**

1st Reader : **Dr KEUNG, Wai Jacky**

2nd Reader : **Prof WANG, Cong**

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Deep Reinforcement Learning in the game of "7 Wonders"

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name: DAWIEANG Phudis Signature: 

Student ID: 55411086 Date: July 12, 2021

Abstract

In recent years, classic abstract games such as Go, chess and Backgammon has become the testing domain for games AI using Monte-Carlo Tree Search. These AIs, however, has been improved significantly using the new novel techniques, which are Deep Reinforcement Learning. In this paper, the author would like to study and apply Deep Reinforcement Learning with the more complex modern board game “7 Wonders”. With large state space, hidden information, asymmetric initial states from different types of boards, trading system, effects and randomness, “7 Wonders” is an interesting game to explore and study on. In this paper, instead of tackles certain problem in each game, the author decides to implement the whole 7 Wonders game and, an OpenAI environment for 7 Wonder. States and Actions space are reduced and encoded to reduce the size, hence reducing the training load. Deep Q-Learning (DQL), consists of Linear Layer and Normalizer, is also used to train itself during play, given only the basic rules. Two neural networks, which are policy network and target network, are used for retrieving a Q-value of each action given the states. Decaying ε-greedy is implemented for exploration and exploitation. Masking is applied to filter out illegal actions. Also, the reward is discounted and distributed at the end of the episode. The result is compared with the rule-based strategy and random based strategy in both 3 players environment and 4 players environment. While the DQL is better than random based strategy, rule based strategy still works better.

Acknowledgement

The author would like to express my gratitude to Prof. CHAN, Antoni Bert for supervising the project and giving guides about the report. Moreover, the author would like to thank City University of Hong Kong for giving me a chance to explore in my own interests subject of Reinforcement Learning.

The author also would like to thank to Mr. David Silver, Richard S. Sutton and Andrew G. Barto for helpful resources about Reinforcement Learning.

I would like to thank my friends who become the beta tester of the program and play the game with me. I also would like to thank my roommate, Mr. Danudej Sukitjavanich, who still supports me during the mental breakdown.

I would like to thank you my belated friend, Mr. Tanawin Puewkhao, who introduces me to the Eurogame 5 years ago, which leads to me knowing 7 Wonders later. It's my pleasure to do this project for you, rest in peace.

Thank you to my parents who still believed in me, when I don't believe in myself.

And lastly, thank you for anyone who picks up this paper in the future.

Table of Contents

Introduction

Background Information

Reinforcement Learning

Machine Learning is a field that receives various interest in the recent years due to its ability to imitate human actions in the speed that faster than human. Reinforcement Learning, however, is one of the machine learning field that different from other machine learning fields such as supervised learning and unsupervised learning. Its main difference is that, instead of learning from human actions, reinforcement learning learns from interaction with itself with the given action-reward policy.

In the past few years, reinforcement learning has proven its success in various fields not only in computer science, but also in real-world situation such as in tradings and robotics.

Motivation

Within recent years, multiple board game has been "solved" by reinforcement learning, such as Go, Backgammon, etc. These board games are mostly ideal in a sense that these games are fully-observable (no hidden information), solitaire (one-player) or two-player, deterministic games. While being able to solve these games are absolutely an achievement in a field of reinforcement learning and an evolution in a past few decades in the field of machine learning, real-world environments are mostly non-ideal.

Non-ideal games are opposite of ideal games that they have hidden information and more than 2 players and stochastic. These games can be found in the certain type of boardgame called Eurogame[1]. There are multiple famous Eurogames that are widely popular, such as Catan, Power Grid and Carcassonne. This makes reinforcement learning in Eurogame interesting and worth exploring. However, most of the paper decided not to use the whole game environment, and solves only some parts of the game. For example, a paper by Xenou, Chalkiadakis and Afantenos has explored the game of Catan[3], which is a fully-observable, multiplayer competitive game with stochastic elements from dice rolls. The paper decides to approach the trading problem in games, and let the existing jSettlers[2] to deal with infrastructure construction.

The author is interested in another Eurogame called 7 Wonders, which are partially-observable, multi-player competitive games with stochastic elements from card shuffles and wonders' assignment. A few years ago, a group of researchers leading by Robilliard, Fonlupt and Teytaud has tried to solve this game using the Monte-Carlo Tree Search[1]. However, they decide to use only cards and discard the boards. The author would like to use Reinforcement Learning technique which is widely used recently called Deep Q-learning, to play a whole game of 7 Wonders including boards and cards.

7 Wonders Game

7 Wonders is a type of 3-7 players card Eurogame, which an extension of boards (or called Wonders) that gives different abilities to the player. Players' objective is to build resources, trade, and create the most developed civilization (which can be determined in the end-game phase with a virtual point called Victory Point or VP in short).

[1]Eurogame are European board game. It is interesting than other common party games such as Monopoly because it requires more critical planning and no player elimination.
[2]jSettlers is Java-based bots to play Catan

The game is splited to three main ages, called ages I, II and III. Later the ages, more impactful the cards are. However, cards in later ages will also require resources retrieved from the earlier ages to be able to play.

Cards are separated into 7 types, which are raw materials, manufactured goods, civilian structures, scientific structures, commercial structures, military structures and guilds. Each type will have a different effect on players' resources and victory points. For example, military strutures will affect the points from the army clashes after the end of each ages. Each card will also have a cost to pay before playing, which are coins or resources, depends on each card.

Before the game is started, Wonders' board will be distributed randomly to each player. Player will be randomly assigned side A or side B of the board, which will have different effects. Amount of cards used depends on how many players played in that game. Some cards will be played only if the certain amount of players reached (for example, 6+ cards will be played only if there are 6-7 players), and guild cards will be kept equal to amount of player +2. Each player will receive 3 coins in the beginning of the game.

The card will be distributed equally to each player randomly. In one turn, a player can select to do 3 possible moves.

1. Play a card
2. Build a stage of his/her own wonder.
3. Discard a card to get 3 coins.

For the first choice, a player can select to pay 2 coins (or 1 if that player has corresponding commercial cards) to neighbor to get 1 resource, in case that the player does not have enough resource to play it.
For the second choice, building a stage will have different effect for the player. For example, Olympia (A) second stage will make a player be able to play one card for free per age.

Olympia (A) board.

The third choice is always possible. However, most of the time it is considered as a disadvantageous move because 3 coins are equal to only 1VP at the end of the game.

After the end of each turn, the sets of card that a player hold will get rotated clockwise in an odd age and anti-clockwise in even age.

At the end of each age, the military clash between each neighbor will occur. Winners will receive points equal to 2age-1 points, losers will lose 1 VP.

At the end of age III, the score will be counted from construction created. Each type of cards will have different way to count VP (for example, VP for scientific construction will equal to summation of the square of amount of each type of construction). The score can be written as

a formula as $VP_{total} = (military\ VP) + \left\lfloor \frac{coins}{3} \right\rfloor + (wonders\ VP) + (civilian\ VP) + (commercial\ VP) + (guild\ VP) + \sum_{p \in \{scientific\}} |p|^2$
The one who has the most VP at the end of the game wins.
The full rules can be seen from the rulebook[3]

Reinforcement Learning in games

Because Eurogames are more related to the real-world situation and required proper strategy to win a game, they are one of the samples used to determine the quality of the AI algorithm implemented.

In the past few years, the technique called Monte Carlo Tree Search is used in various games, such as Catan and Go. However, in 2019, a group of researcher has implemented the Deep Recurrent Reinforcement Learning, using the Q-value to determine next action, to the game of Catan. Moreover, instead of using the traditional experience replay, the algorithm decides to use Long Short-Term Memory (LSTM) to keep previous states' information. The result shows that the Deep Q-learning performs significantly better than the MCTS.

The objective of this project is to use DQL in a game of 7 Wonders described above. The reason that 7 Wonders is selected because it has many problems that Catan does not have, for example, partial information and non-linear reward calculation.

Problem Definition & Scope

The problem proposed in this project is to introduce the Deep Q Learning algorithm to tackle various problem in the game of 7 Wonders, such as partial information, stochasity and being more than 2 players game.

An algorithm introduced will receive the state of the current game as an input for each turn. The input will include the various information, such as amount of resources, amount of certain colors' card, including both player and neighbors. The output will be an action in that turn.

This project will focus on only the main game of 7 Wonders without expansions (Babel, Leaders, etc.). Moreover, the 7 Wonders game used for testing algorithm will be based on the text-based game, not the visual game due to the limited time to train the model.

[3] https://cdn.1j1ju.com/medias/c8/d6/88-7-wonders-rule.pdf

Literature Review

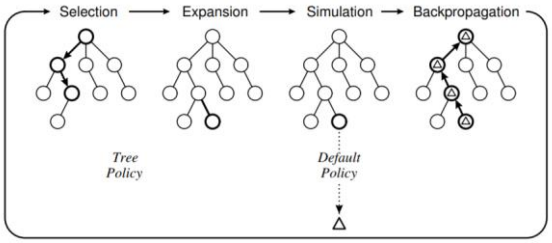
Major alternative to the problem

Rule-based AI

Many AI in games are the rule-based AI. Rule-based AI is an algorithm which based on the human knowledge of the game, then creates strict rules that an AI will follow, without considering external information or planning ahead. With flaws that the rules are based on human knowledge, and the algorithm will only follow the rules regardless of other players' information, rule-based AI performance are limited, and it is normally used in AI game research as a baseline player. For example, the previous research that invented an AI based on MCTS uses an AI with rules that have different priorities as a baseline in the experiment for performance evaluation[1]

Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is an another alternative used in many AI to decide an action. MCTS works by putting a probability of winning on each action, create new nodes if needed, then select to traverse on one branch. Then, the reward will return back as a backpropagation. The MCTS workflow is described in the diagram below.



MCTS algorithm[7]

The MCTS is being used in many games AI, such as the famous AlphaGo[8] , AlphaZero[10] and Catan[2]
In the previous paper that the researcher applied MCTS into 7 Wonders[1], an algorithm uses Upper Confidence Tree (UCT) MCTS with progressive widening and determinization to deal with the partial information, strategic discard and trading strategy. Even though an algorithm works well against the rule-based AI, there are some limitations on this algorithm. While in theory, MCTS will be able to work perfectly, there's a gap between in practice and in theory. The trading strategy is limited with the amount of branches explored, because there are many valid trade choice, and exploring all branches possible will greatly impact the speed of play. Also, the MCTS is more focused on self-built, but less focused on others' construction, which may create some advantages or disadvantages. For example, playing army cards will be less relevant when the player already in a huge leads in army.

TD(λ)

Temporal-Difference (TD) is similar to MCTS, however instead of traversing until the end of the branch then update value with real rewards, TD(0) sampling only one action, then update value with the sampling reward.
TD(λ) traverses with certain depth, then sampling from that depth.

Data Mining

Data Mining is used on the previously played games to find a hidden strategy in the game. From mining the past records of the game played by top human players, strategy can be formulated[4]. However, the limitation is that these strategy cannot be used to create an unsupervised learning AI, which cause the similar problem with the rule-based AI that all the data are based on human plays.

Q-Learning

Q-Learning is an off-policy reinforcement learning algorithm that determine the best action by Q-table lookup. Q-learning has three main steps. First, initialized the Q-table, then explore to the next state, which normally balanced between exploration and exploitation using the ε-greedy policy (with $1-\varepsilon$ chance to exploit/to use the maximum Q-value and ε chance to explore/select next steps randomly), then update the Q-value in Q-table using the Bellman equation written below.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\right)$$

Bellman Equation for updating Q-value[9]

Deep Q-Learning

Q-learning has one main problem, which is that the Q-table can grow exponentially if there are large amount of states and actions. Function approximators are implemented to solve this problem so the Q-table is not required. Deep Q-learning replaces Q-table with neural network for function approximation. Multiple types of neural network have been used, such as Deep Feed Forward, Convolutional Neural Network and Recurrent Neural Network. Recently, Long Short Term Memory (LSTM) has been used effectively because of its ability to remember previous' states information.[3]

Deep Q-learning has been used in many games before. For example, in multiple Atari games[11] where the model receives only the screen pixels as inputs, or FPS games like Doom[12] that dealing the imperfect information using the recurrent neural network

Fictitious Self-Play (FSP)

Fictitious Play are proposed in 1951 by Brown. The methods are focused on playing the game repetitively, and then choose the best response to opponents' average strategy. If the game amount is sufficient, each player's action will lead to the Nash-equilibria in some classes of games. However, Fictitious self-play are rarely used in the past decades comparing to other algorithm based on data mining, rule-based or MCTS because amount of actions are exponentially increased from the game states.

Fictitious Self-Play is an extension to the Fictitious Play, by doing self-play instead and using experience from self-play to evaluate the action.

Other methods (DDQL, Actor-Critic, etc.)

There are also some other extensions from ones mentioned above, such as actor-critic, duelling deep q-networks, A2C, A3C, etc. There are also some examples of using these extensions in game, such as A2C in blood bowl[14]

Preliminary design, solution and system

System overview

The overall will consist of two main parts, the 7 Wonders game in custom OpenAI Gym and the Deep Q network (DQN) algorithm. These two parts will be connected through an action and feedback loop.

The 7 Wonders Game

The 7 Wonders Game is created in Python language. The game is text-based. Each possible action, including all possible trades, will be described as a number for a player to select. At the end of each game, game log will be created and the winner will be declared. These logs data will later be used for the reinforcement learning to improve itself later.

All of the wonders and cards information will be kept as a JSON file.

There are two main types of bots, namely “RuleBasedAI” that uses rules to select the action, and “RandomAI” that random possible action

Bots will be described with the “personality”, which is currently written as a rule-based AI. The personality will be the connection point with the DQN algorithm.

OpenAI Gym

The 7 Wonders Game specified above are available for playing against human, however the game alone does not available for training. The author decides to write a custom OpenAI Gym for 7 Wonders Environment, so it can be used to integrate with other machine learning libraries such as PyTorch.

Deep Q network

The DQN will be used to decide which action will an AI take. DQN consists of two main parts, Deep neural network, which will be used to determine the local Q value for each action, and the action selector, which will select an action with highest Q-value and return the adjusted value back to the deep neural network.

Even though the previous paper has provided a great result using the Long short-term memory (LSTM) for the game of Catan [3] ,the previous paper uses LSTM to evaluate score in each single possible action, and the amount of actions are limited to trading only instead of trading and building. The author decides that within the limited resource, to be able to consider all possible actions, it is better to use the normal Fully Connected Neural Network instead.

There are 2 networks, target network and policy network. Policy network is the network that choose the action, and target network evaluates the policy network’s action.

Q-learning will be explored by using the decaying ε-greedy policy instead of the normal ε-greedy policy. Reward will be given not at the end of each Age, instead it will be given at the end of each game, count as one episode.

Reward will be given not as a points retrieved from the game, but as a ranking. In the 4-player game, reward will be given as [2,1,-1,-2] from ranking 1st to last ranking. The reward will be discounted at the rate of 0.95. Simply said,

$$r_i = (0.95)^{n-i} r_n$$

n = amount of total actions

r_i = reward for the action i

Detailed methodology and implementation

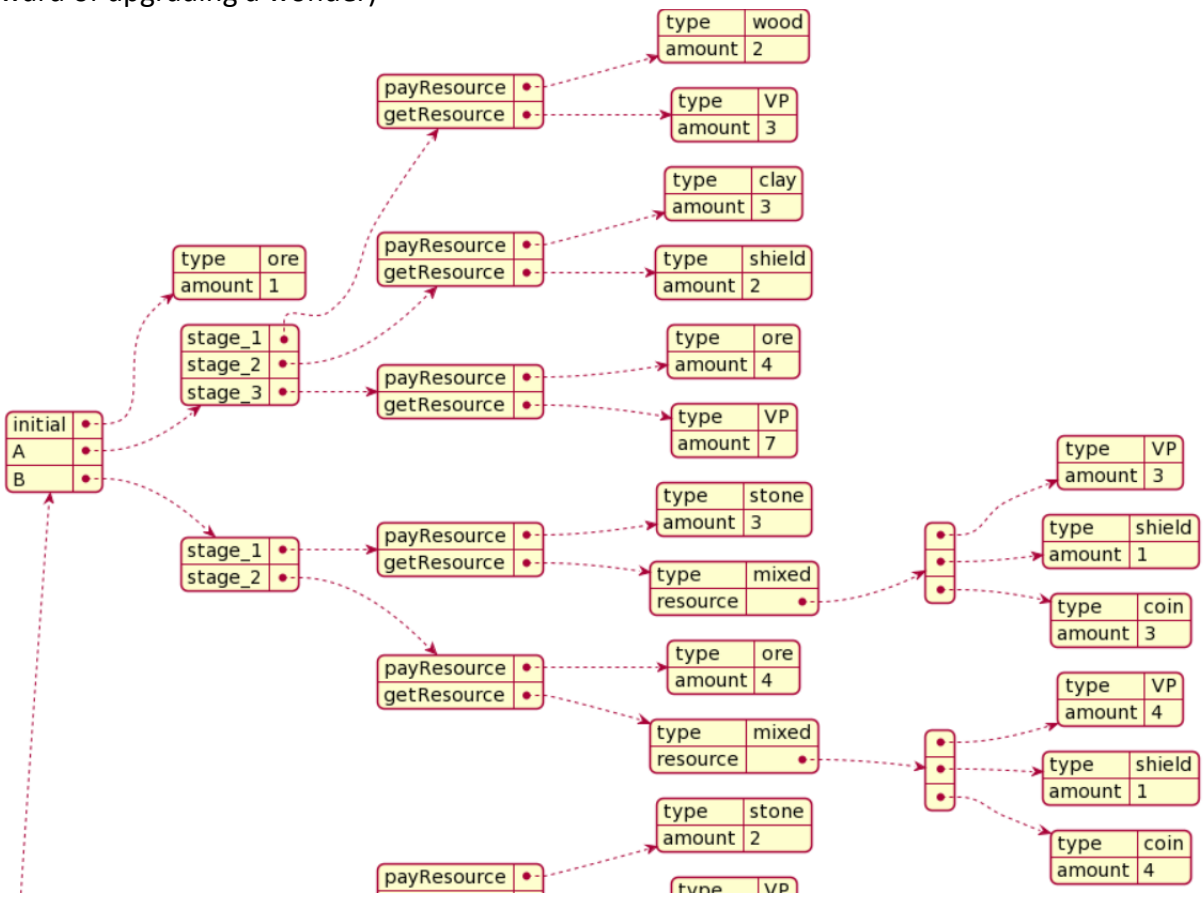
Game Program

The game program contains two main parts, the database and the game system.

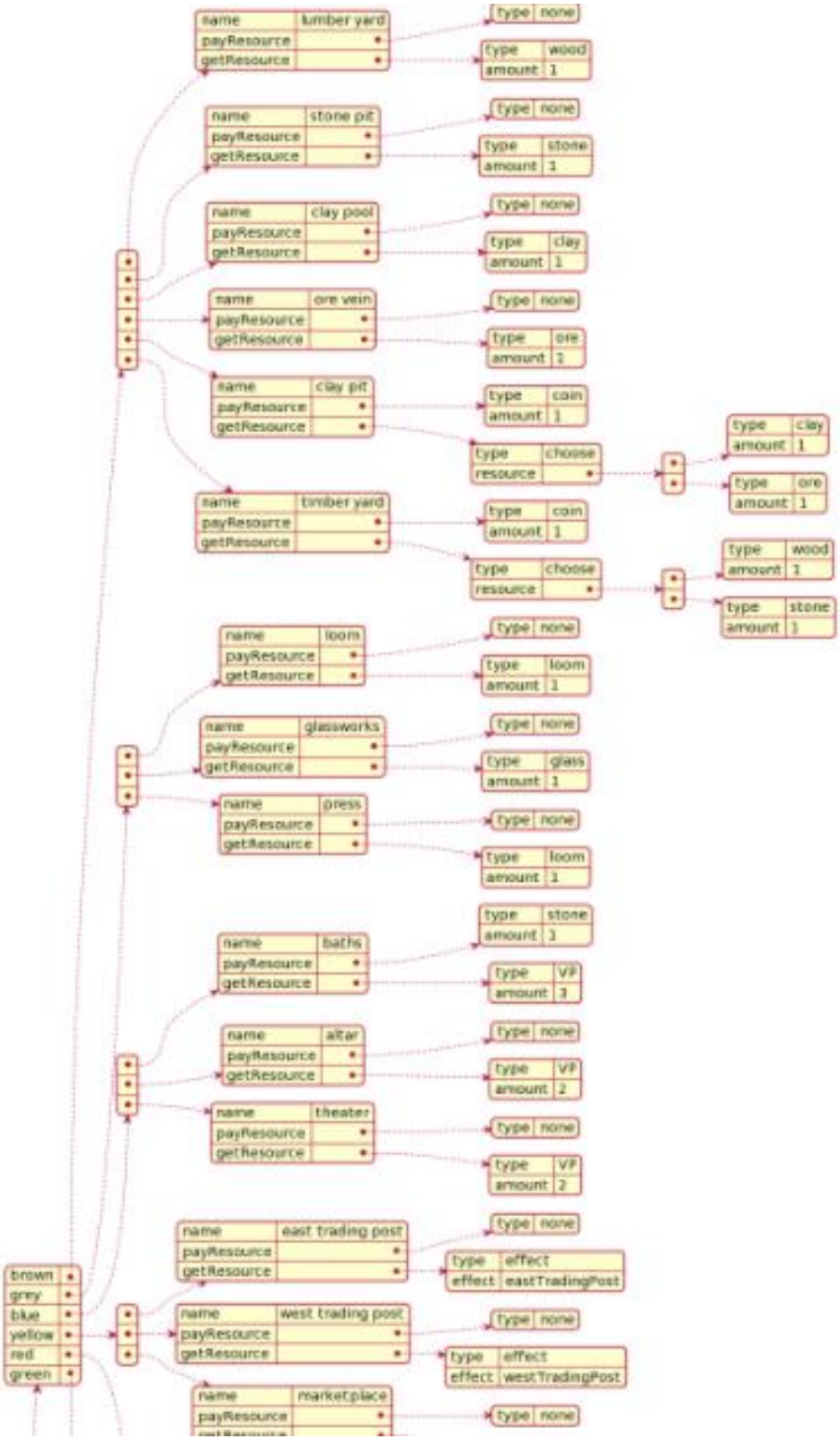
For DB, there are 2 main database kept in JSON file format, which are wonders_list and cards_list.

Wonders are kept as “initial” which tells the given resource with the board, “A” and “B” for each side of the board.

“A” and “B” are divided by stages, and each stages are divided by payResource (Resource needed to pay for upgrading that wonders’ stage) and getResource (Resource retrieved as reward of upgrading a wonder)



Cards are kept separatedly for each ages (“ages_1”, “ages_2” and “ages_3).
For each age, the cards are divided by colors. Inside each colors have multiple cards with name, payResource and getResource.


brown
grey
blue
yellow
red
green
name lumber yard
payResource
getResource
type none
type wood
amount 1
name stone pit
payResource
getResource
type none
type stone
amount 1
name clay pool
payResource
getResource
type none
type clay
amount 1
name ore vein
payResource
getResource
type none
type ore
amount 1
name clay pit
payResource
getResource
type coin
amount 1
type choose
resource
type clay
amount 1
type ore
amount 1
name timber yard
payResource
getResource
type coin
amount 1
type choose
resource
type wood
amount 1
type stone
amount 1
name loom
payResource
getResource
type none
type loom
amount 1
name glassworks
payResource
getResource
type none
type glass
amount 1
name press
payResource
getResource
type none
type loom
amount 1
name baths
payResource
getResource
type stone
amount 1
type VP
amount 3
name altar
payResource
getResource
type none
type VP
amount 2
name theater
payResource
getResource
type none
type VP
amount 2
name east trading post
payResource
getResource
type none
type effect
effect eastTradingPost
name west trading post
payResource
getResource
type none
type effect
effect westTradingPost
name marketplace
payResource
type none

For the game system, there are multiple classes to consider

- Player class who is the agent for our DQL bots and also RuleBasedAI and RandomAI bots
- Resource class
- Wonder class specify each stage's effect, payResource and getResource
- Card
- Personality

Personality

There are three main types of personality that will be used in the project.

- DQNAI : This AI is an AI proposed in this project.
- RandomAI : This AI will select an action randomly from the available option in each turn.
- RuleBasedAI : This AI will play by a certain set of rules with priorities.

RuleBasedAI will try to maximize points using these rules in priority for the age 1 and 2(if rule #1 condition did not met, check rule #2)

1. Upgrading the stage of wonders if it is available and free.
    a. If it's available but it's not free, 50% to pay for stages and 50% to go to next rule
2. Select a card that provides more than one type of resource. If there are multiple cards satisfying this rule, randomly select one.
3. Select a card that provides mixed resource (For example, Clay Pit provides either brick or wood). If there are multiple cards satisfying this rule, randomly select one.
4. Select a card that provide resources that this player does not have yet. If there are multiple cards satisfying this rule, randomly select one.
5. Select military card if it makes this player's military point supercedes the neighbor. If there are multiple cards satisfying this rule, randomly select one.
6. Select science card. If there are multiple cards satisfying this rule, randomly select one.
7. Select VP (civil) card. If there are multiple cards satisfying this rule, randomly select one.
8. Select an action that does not discard card. If there are multiple cards satisfying this rule, randomly select one.
9. Select any remaining playable action.

For age 3, the RuleBasedAI will act similar to MCTS, however it samples only one depth. It will simulate all actions, then select ones that give the most immediate VP.

OpenAI Gym Environment

OpenAI is one of the toolkit for developing and comparing different reinforcement learning technique[13]. OpenAI provides multiple built-in Gym Environment for the training. However, because this project approaches a unique board game that is not as popular as others (such as Go or Backgammon), the custom OpenAI Gym Environment is required.

OpenAI Gym consists of 4 main functions, which are

- Init : Initialize the environment
- Step : Take an action. Return the current state, reward from that state, Boolean specifies if the game is done or not and other extra information.
- Reset : reset the environment at the beginning of the episode

- Render : Render the play in each episode. However, because the game is text-based and not graphical-based, the action can be seen in action log directly.

OpenAI Gym also requires 2 main variables, which are

- Observation_space : All possible states in the game
- Action_space : All possible action in the game.

In 7 Wonders, observation_space is defined as MultiDiscrete consists of

- resourceAmountOwn(11)
  - 10 resources' amount except the battle points are Discrete(5)(*)
  - 1 battle points as Discrete(20)
- colorAmountOwn(7)
  - each color is defined as Discrete(10)
- ownEastTradingCost(7)
  - each trading cost is defined as Discrete(2)
- ownWestTradingCost(7)
  - same as ownEastTradingCost
- ownCoin(1)
  - Discrete(100)
- resourceAmountLeft(11)
- colorAmountLeft(7)
- resourceAmountRight(11)
- colorAmountRight(7)
  - Same as Own, but for left and right neighbor
- AgeNumber(1)
  - Discrete(3) because there are 3 possible ages.

(*) Discrete(n) is a Gym class specifies that there are n possible values from 0 to n-1

Action space are defined by 2 values

- CardCode : Discrete(75). There are 75 unique cards in this game, each card will have its own CardCode from 0-74
- ActionCode : Discrete(4). There are 4 unique actions can be done per card
  - 0 : Play normally and pay the cost to neighbor (if needed)
  - 1 : Play without paying the cost (by using wonders' effect, for example Olympia side A 2nd stage let player build one card free per one age)
  - 2 : Discard the card for 3 coins. This option is always possible.
  - 3 : Use a card to build stage on the players' wonder.

However, while multiDiscrete observation_space can be passed directly as an input layer in neural network, it is harder to have 2 different output layers. The author decides to encode from (CardCode, ActionCode) to single action represents by number between 0-299, making action_space as a Discrete(300). The action number can be converted easily by

$$actionNum = f(cardCode, actionCode) = 4 * cardCode + actionCode$$

There are some other functions needed to be designed, such as

- actionToCode/codeToAction : Interchange the action and the actionCode to be used in the DQL as mentioned before.

- LegalAction : Return the legal action of the player. This will be useful for action masking, which will be explained in the DQL session.
- stateGenerator : Generate the current state as observation_space.

Deep Q-learning

Deep Q-learning is implemented in Python using PyTorch for creating the Q-network. The Q-network is used to mapped between the state S and each Q-value Q(S,a). It is implemented as a neural network, consisting of 1 input layers, 5 hidden layers and 1 output layers

1. Linear(70,140)
2. LayerNorm(140)
3. Linear(140,200)
4. LayerNorm(200)
5. Linear(200,300)

However, the output layer V_o cannot be used directly because most of the actions are illegal actions (Using cards that are not in hand, play a card that a player does not have enough coin to pay for additional resources or there are no neighbor with those type of additional resources, play a card for free while do not have an effect that can play it, upgrade wonders when the wonder is already fully upgraded, etc)

LegalAction function mentioned above sends an array L consists of action number that is legal. This array generates the masking vector M by setting 1 to index that is valid, and 0 to index that is invalid.

$$M[i] = \begin{cases} 1; i \in L \\ 0\ ; otherwise \end{cases}$$

After that, the action is decided depends on the decaying ε-greedy policy. Random number r will be generated between 0 to 1, and then the action a is decided by following formula

$$a = \begin{cases} a_g\ ; r < \varepsilon \\ a_r\ ; otherwise \end{cases}$$

ε is the threshold to decide which action to do, which can be calculated by

$$\varepsilon = \varepsilon_t + \frac{\varepsilon_s - \varepsilon_t}{e^{(-\frac{episode}{\varepsilon_d})}}$$

when ε_t represents terminal threshold when the episode $\to \infty$, ε_s represents initial threshold and ε_d represents the decaying rate of ε

Decaying ε-greedy policy is used here to prioritize exploration in the first few episodes, and then prioritize exploitation on later episodes.

Greedy action a_g can be retrieved by taking Hadamard product between output layer V_o and masking vector M, normalize the value to between 0-1, then find the action number that gives the maximum value.

$$a_g = \underset{a \in L}{\operatorname{argmax}}(V_o \circ M)(a)$$

Random action a_r can be retieved by randomly select on of the possible action.

$$a_r = \underset{a \in L}{\operatorname{rand}}\, a$$

In each action, if payment is required, the payment will be made greedily that the player will have to pay the least amount of money.

Reward will be given not as a points retrieved from the game, but as a point for getting certain ranking. In the 4-player game, reward will be given as [2,1,-1,-2] from ranking 1st to last ranking. In the 3-player game, reward will be given as [2,1,-1] instead.

Moreover, the reward will be given after the end of each episode, not a step or age. The reason behind this planning is that, in 7 Wonders, some cards have less value in game, but can be accumulated to generate large values in the end. For example, science card that gives science resources. Wheel or tablet alone give only 1 VP, however, 4 of tablets give 16 points.

Discounted reward model is used for the reward distribution for each certain (state,action) pair

$$r_{s,i} = (k)^{n-i} r_n$$

n = amount of total actions

$r_{s,i}$ = reward for the action i in state s

k = discounted rate

For policy network evaluation and training, another network called target network is created. Target network will generate Q-value, and loss will calculate batch by batch using the SmoothL1/Huber loss. These loss will be used to adjust the policy network.

Q-learning is off-policy learning because it uses target network to evaluate policy network, not policy network to evaluate itself.

Problems addressed in 7 Wonders

There are many problems in 7 Wonders that need to be considered. For example,

- Large state sizes. Keeping all cards currently played with effects created large state sizes.
    - Keeping all the card as a state can create a very large amount of states, with the amount of possible $\left(2^{75 \text{ cards} + (7 \text{ Wonders} * 3 \text{ average stages per wonder})} * 50\ maximum\ coins\right)^{7\ players} \sim 1E203$ number of possible states.
    - The author decides to reduce number of states by using amount of resources, amount of color cards, trading price and age instead to reduce number of states.
    - Moreover, instead of keeping states of all members, the author kept states of only a player and their neighbors only. While this is definitely a loss of game information, the author presumes that other players' state do not help much more than neighbors' state. A player would have less chance to play a card to block enemies' action that will receive his/her hand in 5 turns than play a card to prevent his/her neighbor to play it.
    - The amount of new possible states are $5^{10} * 20 * 10^7 * 2^7 * 2^7 * 100 * 5^{10} * 20 * 10^7 * 5^{10} * 20 * 10^7 * 3 \sim 1E52$, which, even though it reduces a lot from previous states, is still a large number. Which is why DQN is implemented to replace Q-table look up.
- Large action size.
    - If we consider the amount of possible actions directly, there are 75 cards, 7 Wonders, 3 Ages, 3 actions (play,discard,upgrade wonder). These are still in an

acceptable scale, however, considering the amount of possible payment possible, the amount of actions will escalate very quickly (in some cases, there are estimated 30-40 possible payments to be made)
    - The author decides to reduce amount of action by encoding the (card,action) pair to 300 possible action codes. Moreover, after selecting the action, the payment is done greedily to reduce the size of the action state.
- Hidden information. A player cannot know what's in enemy's hand.
    - While hidden information still exists in this situation, the resource, coin and color can be seen directly from states.
- Weak actions. Discard for 3 coins is the weak action, and many episode will be lost on this weak actions.
    - Even though the past paper in 2014 decides to consider discard as a last possible option to do[1], the author considers that discarding can be a valid action sometimes. For example, if player's next neighbor has 3 wheels, and player has wheel on hand but cannot play it, player can decide to discard it so that player's neighbor will not get 7 more VP from 1 card ($4^2 - 3^2 = 7$)

Experiments and results

Experiment

Experiment are done in 2 different batchs.

First batch for 3 players game consists of 2 game environment

- 1 DQNAI, 1 RuleBasedAI and 1 RandomAI
- 1 DQNAI and 2 RandomAI

Second batch for 4 players game consists of 3 game environment

- 1 DQNAI, 2 RuleBasedAI and 1 RandomAI
- 1 DQNAI, 1 RuleBasedAI and 2 RandomAI
- 1 DQNAI and 3 RandomAI

The model will be exported every 200 episodes. One episode ends when a game ends.

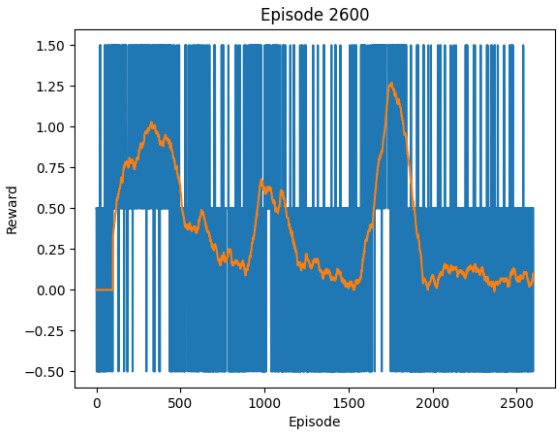
In every experiment, there are multiple fixed value for thresholds and network updates

- Decaying reward rate k = 0.95
- Terminal threshold ε_t = 0.05
- Initial threshold ε_s = 0.9
- The ε decaying rate ε_d = 2500
- Target Network update every 10 episodes
- Max Episode = 2600
- Rewards mentioned above for each game.
- Batch size for loss function = 128
- Gamma for expected Q value calculation = 0.99

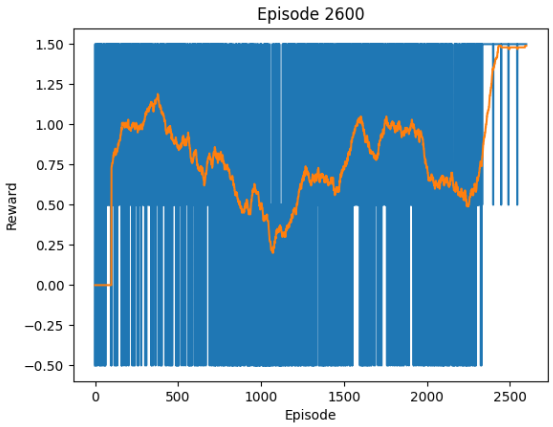
Results

The result is represented in graph, consist of each episode's reward (Blue line) and Simple Moving Average of 100 Episodes (Yellow Line)

3 players game :

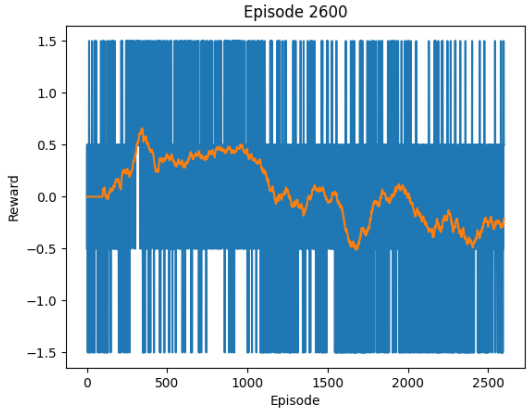


1 random + 1 rule + 1 DQN

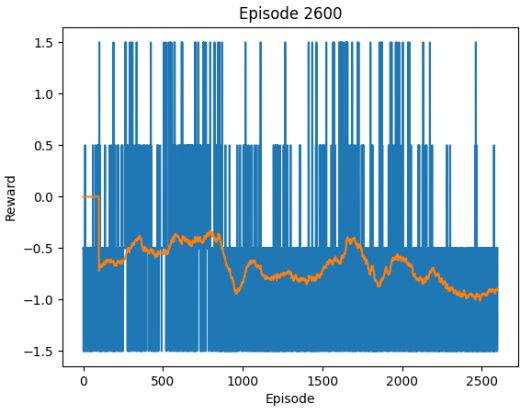


2 random + 1 DQN

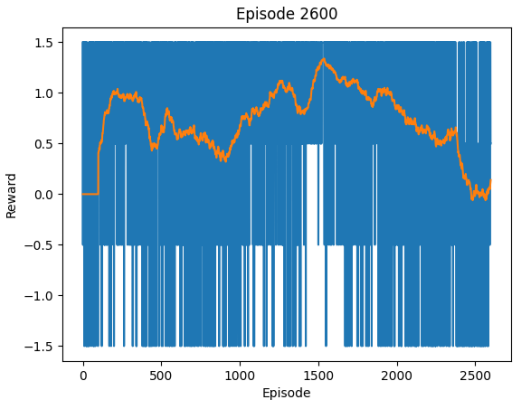
4 Players Game



2 random + 1 rule+ 1 DQN



1 random + 2 rule+ 1 DQN



3 random + 1 DQN

From the result, it can be seen that the DQN agent performs well against the RandomAI agent, as the Moving Average of the Reward are mostly between 0.5 to 1.5 (1st or 2nd place) However, it does not perform well against the RuleBasedAI agent, as the moving average is significantly reduced when RuleBasedAI is introduced.

Concluding Text

Critical Review

From the result of the testing, it can be seen that the current DQLAI algorithm that this project proposed, even though using only normal fully connected neural network, performs better than the RandomAI. However, with current parameter tuning, Rule-based AI performs in much higher efficiency than DQLAI right now. There are many ways that the author thinks this DQLAI can be improved, for example:

- Using better neural network, such as LSTM. Our current network performance are not good, and with right parameter tuning and better architecture, the author believes that higher performance can be achieved.
- Change the state/action space. While the current state/action space are small and practical on lightweight machine like author's current laptop, scaling down states and actions also mean that some of the information are lost. For example, ones can decide to extend state so that it also keeps previous moves from the enemy.
- Teaching model directly about the illegal action. Currently, the masking layer is applied to the output layer of the neural network, not the hidden layer. Within author's current knowledge, it is harder to use masking layer as a function similar to how activation function layer is used (Relu, tanh, etc)
- Using graphic card for faster training. The current network uses CPU for training because author's AMD graphic card does not support CUDA.

Also, from author's perspective, the RuleBasedAI is not exactly pure RuleBaseAI, because its Age 3 action depends on the maximum intermediate return (VP) instead of using set of priority rules in Age 1-2. It is interesting in a sense of how does RuleBasedAI combines with reinforcement-learning-like observation can give a very good result. Maybe in the future, this can be used for early episodes training when policy network and target network are not stable yet, then change to full DQN after.

From some results that authors speculates using the game log, RuleBasedAI does not really play civils card. Instead, it played lots of Science card or Guild card in age 3. This may refers to these cards' ability to gain huge amount of VP in short time.

On the bright side, this project has introduced the custom OpenAI Gym environment on 7 Wonders, which no one has done before. While in this project, the DQNAI does not work really well, the author hopes that in the future, when someone who has more knowledges about these algorithm, one can apply new algorithms or new designs to this custom environment and explore moreover into the non-ideal board games. The author plans to clean up the code, and submit this environment to RLCard, which is a famous open-source toolkit for reinforcement learning in card games.

There are many problems arise during this project. Most of them comes from the author's lack of knowledge, and only overflowing passion in the subject. Before this project starts, the author thought that he knows lots of knowledge, then realized later that he's only on the top of the Dunning-Kruger's peak. Full with confidence, but lacks of knowledge.

There are also technical problems arised. For example, OpenAI does not support Windows OS. The author decides to uses Linux, which its command language that the author does not know much about, comparing to other language in Windows. Moreover, some

packages only existed in pip environment, and some others existed in conda. Combining these two together is a hard task, which the author decides to find some alternatives instead.

Moreover, with lacks of experience for Object-Oriented Programming despite all courses the author has studied before, coding the whole game is very hard and daunting experience. Before the project starts, the author found the online code of 7 Wonders, however, it is not completed, so all coding must be started from scratch. Then, OpenAI Gym Environment has to be created for training. After all of these things are done, there are not so much time to consider and test about the real part of the project, which is the DQN.

Summary of Achievements

In this paper, the author has introduced the completed 7 Wonders game text-based, written on pure Python. Moreover, the custom OpenAI Gym is created for others to use in the future project. Moreover, a simple DQN algorithm AI to play a game is introduced in this project, even though its performance cannot surpass RuleBasedAI.

Future Plans/Suggestions to extensions

There are many extensions can be made from this project. For example, trying other Reinforcement Learning model, new states/action spaces format,etc. Ones can also decide to extend the RuleBasedAI to apply in general cases, such as in other EuroGames like Power Grid or Catan.

Parameter tuning and testing for games is also an interesting subject to go in. How much epsilon parameters and batch_size should be set, etc.

Monthly Logs

Month	Content
October	The progress of the project (on 30 October) is described below • Creation of the Seven Wonders game in Python: 75%. This is late from the timetable by one week that the standalone game should already be finished on around 25th October. The card/resources database can be retrieved from GitHub, however, the system needs to be implemented. Currently, the only system that is needed to be implemented is the trading/buying resources system and the logfile generation of each game. The game itself should be finished in 1 weeks' time from now (Before 6th of November) • Policy drafting: 20%. In each turn, a player can select 3 different actions, which are playing a card, upgrading a personal board/wonder and discard for 3 in-game money. Moreover, playing a card sometimes will require a player to buy resource from the neighbours. Each action will try to maximize Q value approximated by the deep neural network.
November	The progress of the project (November) is described below • Creation of the Seven Wonders game in Python: 20%. This is reduced significantly due to the fact that the game system provided does not function properly. Also, the database does not consider the free-card buy (For example, buying Statue in Age II makes the player be able to buy Gardens in Age III for free.) That means the game system, the database, tradings, and logs need to be revamped. • Policy drafting: 30%. Additional from the last month, reward should be distributed not as a Victory Point directly, but should be distributed as a ranking, because getting high points in the game does not mean you win the game, and I want to prioritize on winning more than getting more points.
December	The progress of the project (December) is described below • Creation of the Seven Wonders game in Python: 40%. The database of the cards and wonders are kept in the JSON file. • Policy drafting: 60%. Deep Q-learning will be used for selecting the next move. Action will be done as a card selection and state (play, discard, upgrade)
January	The progress of the project (January) is described below • Creation of the Seven Wonders game in Python: 60%. The initialization (reading JSON file, preparing card for each ages) is complete. • Policy drafting: 60%. • Environment Setup : 20%. OpenAI Gym will be used for setting up the custom environment.

February	The progress of the project (February) is described below • Creation of the Seven Wonders game in Python: 70%. Card playing eligibility is 75% completed with a few bug cases. • Policy drafting: 70%. • Environment Setup: 40%.
March	The progress of the project (March) is described below • Creation of the Seven Wonders game in Python: 80%. The StupidBot (select first choice, if there are no choice will discard) is implemented for the program for testing. • Policy drafting: 70%. • Environment Setup: 40%. • Neural Network Architecture: 20%. LSTM layers is used to predict q-value and adjust the weight inside itself
April	The progress of the project (April) is described below • Creation of the Seven Wonders game in Python: 90%. The possible choices expand from buying only from any guy separated into buying from left neighbour and right neighbour because buying from winning players are worse than buying from losing players. • Policy drafting: 70%. • Environment Setup: 50%. • Neural Network Architecture: 20%
May	The progress of the project (May) is described below • Creation of the Seven Wonders game in Python: 95%. Tiny bugs about checking if the resource is enough or not, VP calculation complete, program is almost useable now. • Policy drafting: 70%. • Environment Setup: 70%. • Neural Network Architecture: 60%. Keras is used for implementation.
June	The progress of the project (June) is described below • Creation of the Seven Wonders game in Python: 100% • Policy drafting: 70%.

	• Environment Setup: 85%. After the Seven Wonders game is finished, OpenAI Gym is used for basic formatting of training • Neural Network Architecture: 70%. The author decides to change from Keras to Pytorch instead, because of the dependencies issues
July	The progress of the project (June) is described below • Creation of the Seven Wonders game in Python: 100% • Policy drafting: 100%. • Environment Setup: 100%. • Neural Network Architecture: 100%. • Testing : 100%

References

[1] Robilliard, D., Fonlupt, C., & Teytaud, F. (2014). Monte-Carlo Tree Search for the Game of "7 Wonders". *Communications in Computer and Information Science Computer Games,* 64-77. doi:10.1007/978-3-319-14923-3_5

[2] Szita, I., Chaslot, G., & Spronck, P. (2010). Monte-Carlo Tree Search in Settlers of Catan. *Lecture Notes in Computer Science Advances in Computer Games,* 21-32. doi:10.1007/978-3-642-12993-3_3

[3] Xenou, K., Chalkiadakis, G., & Afantenos, S. (2019). Deep Reinforcement Learning in Strategic Board Game Environments. *Multi-Agent Systems Lecture Notes in Computer Science,* 233-248. doi:10.1007/978-3-030-14174-5_16

[4] Joaquim A., Gabriel P., Julia A., Rafael V. & Lana R. (2019). Data mining 7 Wonders, the board game. Paper presented in Proceedings of SBGames 2019, Rio de Janeiro, Brazil

[5] Shao, Kun & Tang, Zhentao & Zhu, Yuanheng & Li, Nannan & Zhao, Dongbin. (2019). A Survey of Deep Reinforcement Learning in Video Games.

[6] Niklaus, J., Alberti, M., Pondenkandath, V., Ingold, R., & Liwicki, M. (2019). Survey of Artificial Intelligence for Card Games and Its Application to the Swiss Game Jass. *2019 6th Swiss Conference on Data Science (SDS)*. doi:10.1109/sds.2019.00-12

[7] Browne, C. et al. A survey of Monte-Carlo tree search methods. IEEE Trans. Comput. Intell. AI in Games 4, 1–43 (2012)

[8] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[9] Silver, D. (2015). *Model-free Control*. *COMPM050 Reinforcement Learning*. London; University College London.

[10] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, *362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. ArXiv:1312.5602 [Cs]. http://arxiv.org/abs/1312.5602

[12] Adil Khan, Jiang, F., Liu, S., & Omara, I. (2019). Playing a FPS Doom Video Game with Deep Visual Reinforcement Learning. Automatic Control and Computer Sciences, 53(3), 214–222. https://doi.org/10.3103/s0146411619030052

[13] Brockman, G. et al., 2016. Openai gym. arXiv preprint arXiv:1606.01540.

[14] *Reinforcement Learning II: A2C*. Bot Bowl. (n.d.). https://njustesen.github.io/ffai/a2c.html.

Appendix

Github Repository for 7 Wonders Game in text (for playing)
https://github.com/MirrorCraze/7WondersText

Github Repository for 7 Wonders OpenAI environment (for training) + Graph photos
https://github.com/MirrorCraze/7WondersEnv