Jarno Hyrkäs

REINFORCEMENT LEARNING IN A TURN-BASED STRATEGY GAME

Kajaanin ammattikorkeakoulu
Bachelor of Business Administration, Information Technology
Fall 2015

TIIVISTELMÄ

Kirjoittaja: Hyrkäs Jarno Juha Tapio

Otsikko: Vahvistusoppiminen vuoropohjaisessa strategiapelissä

Tutkinto: Tradenomi, tietojenkäsittely.

Tämän opinnäytetyön tarkoituksena on tutkia vahvistusoppimista vuoropohjaisessa strategiapelissä. Teoksen teoreettisen osuuden alussa kerrotaan mitä tekoälyllä tarkoitetaan. Myöhemmissä kappaleissa perehdytään koneoppimiseen, vahvistusoppimiseen ja Q-oppimiseen. Teoreettisen tutkimuksen pohjalta opinnäytetyössä rakennetaan Q-oppimista hyödyntävä tekoäly.

Ohjelmointi suoritetaan Unity3D-ohjelmistolla käyttäen c#-ohjelmointikieltä. Tekoäly rakennetaan mahdollisimman helposti muokattavaksi, jotta sitä voi käyttää useassa eri sovelluksessa. Valmiin mallin pohjalta rakennetaan King of Thule-peliin oma tekoälysovellus. Siinä käytettävät pelitilat ja –toiminnot ovat dokumentoituna käytännön osiossa.

Tekoälyä opetetaan peluuttamalla sitä itseään vastaan. Opetusvaiheen tulokset ja niiden perusteella tehdyt johtopäätökset esitellään opinnäytetyön lopuksi.

ABSTRACT

Author: Hyrkäs Jarno Juha Tapio

Title of the Publication: Reinforcement Learning in Turn-based Strategy Game

Degree Title: Bachelor of Business Administration, Business Information Technology

The goal of this Batchelor's thesis was to study reinforcement learning in a turn based strategy game. The theoretical examination or the study includes chapters from explaining what is AI to in-depth analysis of Q-learning, which is the learning method used in the application.

A general reinforcement learning template was made based on the theoretical examination. The code was done with C# Unity3D and the structure is explained in the application section. The code template was used to create AI for a based strategy game. The game is called King of Thule and the Bachelor's thesis has a chapter which explains how the game is played. Another chapter explains what game states and actions were used for the game AI.

The finished AI was trained by making AI agents play each other in a learning period. Data from the period was recorded and analysed.

TABLE OF CONTENTS

List of Symbols

1. INTRODUCTION TO AI 1

2. GAME ENVIRONMENT 2

3. LEARNING 4

- 3.1. Machine learning 5
- 3.2. Unsupervised Learning 5
- 3.3. Supervised Learning 6

4. REINFORCEMENT LEARNING 7

- 4.1. Blame attribution problem 8
- 4.2. Exploration 8
- 4.3. Convergence 8
- 4.4. Approximation 9
- 4.5. Application 9
- 4.6. Q-learning 10

5. DECISION TREES 11

6. CORRELATION 12

7. CODE STRUCTURE 13

- 7.1. Sub state 13
- 7.2. State 13
- 7.3. Action 14
- 7.4. Action State 14
- 7.5. Reinforcement learning 14
- 7.6. Reinforcement handler 15

8. KING OF THULE 16

9. GAME SPECIFIC AI DEMANDS 19

9.1. Adversarial solution 19
9.2. Stochastic solution 19
9.3. Action possibility 20
9.4. Continuing action 20
9.5. Free actions 20
9.6. Learning efficiency 21
9.7. King of Thule actions 23
9.8. King of Thule sub states 24

10.RESULTS 26

11.REFLECTION 29

11.1. Game simulation speed 29
11.2. Game state initialization 29
11.3. Performance 29
11.4. Future applications 30

REFERENCES

LIST OF SYMBOLS

a	is action in a game,
r	is reinforcement value of a state,
s	is game state,
α	is learning rate for Q-learning. Value between 0 and 1,
γ	is discount factor for Q-learning. Value between 0 and 1,
π	is policy. Determines how to choose an action,
Passive learning	has fixed policy,
Active learning	does not have fixed policy

1. INTRODUCTION TO AI

Coppin (2004, 4) defines artificial intelligence (AI) as "*study of systems that act in a way that to any observer would appear to be intelligent*". He notes that an artificial intelligence also includes techniques that are used to solve simple problems and therefore gives another definition: "*Artificial Intelligence involves using methods based on the intelligent behavior of humans and other animals to solve complex problems.*"

An artificial intelligence is used for making computers perform thinking tasks that humans and animals are capable of. Computers are already good at arithmetic, sorting and searching. However, they have problems with image recognition, speaking languages, decision making and being creative. (Millington & Funge 2009, 4).

Millington and Funge (2009, 4) list three different AI domains. Philosophy approach tries to understand the nature of thought and intelligence. Its goal is to create a software model of thinking. The second domain is about the psychology of understanding the human brain and mental processes. The final domain is engineering where people try to create algorithms to perform human-like tasks. Russell & Norvig (2010, 2) divided AI into four groups: Thinking humanly, thinking rationally acting humanly and action rationally. The rational system chooses the correct answer if it knows what it is. The human approach is partly empirical science requiring observations and hypotheses about human behaviour.

Game AI can be defined as a code structure that gives computer controlled entities the ability to make smart decisions. Schwab (2009, 3) lists different applications of game AI. According to him game AI can be understood as agent behaviour. He adds that many people think game AI is primarily a set of low level actions, such as animation selection in a given game state. Lastly Schwab notes that movement and collision algorithms can also be included in game AI.

2. GAME ENVIRONMENT

Game environment dictates which kind of AI system should be used. Russell & Norvig (2010, 42 - 44) present four defining variable pairs (Table 1.)

Table 1. Game environment variables

Fully Observable	Partly observable
Deterministic	Stochastic
Discrete	Continuous
Known	Unknown
Non adversarial	Adversarial

In an observable environment AI does not have to estimate the current game state. This means that it can make the best choice it knows for that situation. Chess is a good example of a game where both players have complete knowledge of the game. In partly observable environment the player does not know everything about the opponent or the environment. Poker is a good example of a partly observable game. In poker the player does not know which cards the opponent has and which cards are coming from the deck.

Deterministic environment always uses the same result given same action in the same state. In chess moving a piece from one place to another always has the same consequences provided the board state is the same. Stochastic game has randomness in it. Performing an action in stochastic game leads to different states at different times. Any game with dice in it is considered a stochastic game. Rolling the dice gives a random value between one and six.

Discrete and continuous game differ from each other with the way time and the state of the environment are handled. In addition, agent observations and actions are different in continuous and discrete games. In simple terms a discrete game has finite states. In chess there is a limited amount of ways the pieces can be placed on the board. Driving a car has continuous state as well as continuous time. (Russell & Norvig 2010, 44).

In a known system the agent has knowledge of what will happen if an action is performed. This does not mean that the system has to be deterministic or fully observable. When playing poker one does not need to know what cards are coming. It is enough to know how to play the game. (Russell & Norvig 2010, 44).

Non adversarial game such as solitaire is played without opponents. In an adversarial game the decision making requires AI to predict what the opponent is doing in the next phase. It chooses the alternative that maximises its next state value assuming that the opponent will perform an action that minimizes it. (Norvig & Thrun 2015)

3. LEARNING

Shi (2011, 18) describes learning as the "*process of acquiring knowledge, gaining experience, improving performance, discovering rules and adapted to environments*". There are different types of learning. It becomes problematic when it is used to predict something. Daumé (2012, 10) lists four canonical problems in prediction.

- Regression: Predicting a real value based on past experience.
- Binary Classification: Predicting yes or no answer on past experience.
- Multiclass classification: Classifying input into one of many groups.
- Ranking: Ordering objects by their relevance.

Learning can be done either offline or online. The online method is done during the time the game is played and offline is done before the end user plays the game. Online learning allows the game to adapt to the player's actions but has problems with predictability and testing. Most Game AIs are built using offline learning methods. (Millington & Funge 2009, 579.)

There are two sides when choosing a representation scheme. The richer it is, the better it will do in future problem solving. It also follows that the richer the representation is, the more difficult it is to learn. Rich representation requires a large data amount to learn and often different hypotheses are consistent with same data (Poole & Mackwort 2010, 7.1).

3.1. Machine learning

According to Barber (2010, 291) machine learning is a research related to automated large-scale data analysis. He adds that the long term goal for many is to produce models and algorithms that can process information. Russell & Norvig (2010, 2) write that machine learning is used to adapt to the new environment and to detect and extrapolate patterns. Learned data can be used to construct a model that predicts action outcomes based on previous learning (Bell 2014, 3). A basic learning system is illustrated in Figure 1.

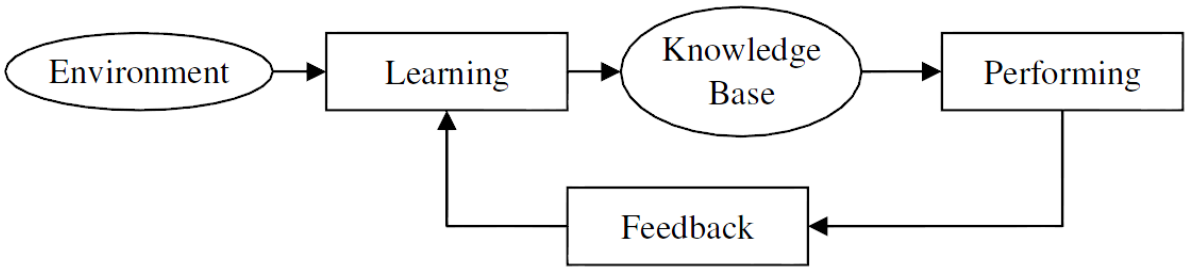

Figure 1. Simple model of learning (Ethem. 2010, 448)

Shi (2011, 18) writes that machine learning enables machines to automatically acquire knowledge and intelligence. He continues that machine learning can also be used for uncovering principles and secrets of human thinking and learning. Bell (2014, 4 - 9) lists modern applications of machine learning: Junk mail spam detection, voice recognition, stock trading, robotics, medicine and health care, advertising, retail and E-Commerce and gaming analytics. Bell (2014, 3) explains that most machine learning algorithms fall into one of two categories: unsupervised or supervised learning.

3.2. Unsupervised Learning

Ethem (2010, 11) writes that the goal for unsupervised learning is to find regularities in the input data. According to Bell (2014, 3) there are no right or wrong answers in unsupervised learning. He adds that the algorithm is merely run and afterwards the results show what kind of patterns and outcomes occur. Unsupervised learning has more to do with data mining than actual learning (Bell 2014, 3).

3.3. Supervised Learning

Supervised learning means that the system is taught with training data (Bell 2014, 3). The teaching phase needs to have input and corresponding output data. The data is used to modify the system so that it produces right outputs with the training data inputs. After the training phase the system is tested with new data to make sure it works properly (Barber 2010, 292). Poole & Mackwort (2010, 7.1) note that if input data is discrete, the process is called classification. They add that if the data is continuous it is referred as regression.

Bell (2014, 3) notes that supervised learning has a bias-variance dilemma. If the learning model has a high bias it learns the training set very well, but works poorly with different training sets. However if the model has a high variance it does not perform as well with the original training data as the high bias model. Instead it works better with new data since it is less affected by noise.

4. REINFORCEMENT LEARNING

In reinforcement learning new knowledge is gained through interaction with the environment. Shi (2011, 362) explains that reinforcement learning maps states to actions. This means that the result maximizes the scalar reward of the reinforcement signal. The agent does not need to know what happens after any choice it makes. Instead it takes an action and learns from the resulting state (Figure 2.) Poole & Mackwort (2010, 11.3) stress that even though agent does not need to know what happens after an action it does need to understand that the result applies only at the starting state. An action may be beneficial in one game state and harmful in another.

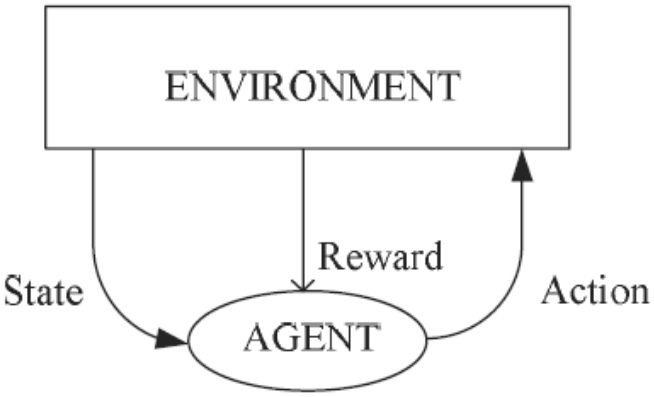

Figure 2. Reinforcement learning principle (Ethem. 2010, 448)

Millington and Funge (2009, 631) note that reinforcement learning includes a number of techniques that learn from experience. They add that reinforcement learning is a hot topic in games and that will be used more in the next-generation gameplay. Russell & Norvig (2010, 831) write that reinforcement learning is many times the only way to train a program to perform well in complex situations. Those situations include game playing and robotics (Russell & Norvig 2010, 831).

4.1. Blame attribution problem

Blame attribution problem is the problem of determining what has caused change in the current state. The change may have been caused by a resent action but it also may have resulted from an earlier action that has delay. It is also possible that the change did not happen due to any single one action but rather combinations of certain actions. (Poole & Mackwort 2010, 11.3).

4.2. Exploration

In reinforcement learning knowledge is attained by taking actions and examining the following states. How actions are chosen at any given state is determined by the exploration strategy also known as the policy (Millington and Funge 2009, 634). In a passive learning the policy is fixed whereas in active learning it changes (Russell & Norvig 2010, 831).

If the agent always selects the best known option it is called a greedy agent. It is not reliable since after it has reached the end goal it will stop trying to find new paths and continues to use the one it knows. In order to make sure the optimal path is found randomness needs to be added for the choosing process. If randomness is maximized there is a risk that the exploration gets stuck and it never finds the goal. A simple solution is to choose the best alternative by a certain probability and the random one with remaining probability. While this would lead to an optimal path it can still be very slow. A better way to adjust exploration is to push for routes that have not been tried very often and avoid paths that are likely to have low utility. (Russell and Norvig 2010, 840 – 841.)

4.3. Convergence

Explorations based learning methods require iterations to map the actions and the states. After enough iterations calculated values no longer change which means that the algorithm has reached an end. Millington and Funge (2009, 634) note that reaching an end point in practical application often takes a vast amount of iterations and therefore iteration values tend to be used before they have settled completely.

4.4. Approximation

Reinforcement learning requires that the game is a set of linked states and each game state is changed by performing an action. Many modern games are continuous or complex. In order to use reinforcement learning in a continuous game discrete approximations need to be made (Millington and Funge 2009, 624). Approximations can be used to decimal numbers and integers such as position, health and ammo count. Russell and Norvig (2010, 846) note that approximating states does not only reduce the number of states but also allows the agent to generalize information. This means that gained knowledge from visited states can be used in in other states in some capacity.

Even after variable approximation the game might have a vast state space which means that it requires a lot of simulation time. The necessary iteration amount can be reduced by guiding the learning process. Millington and Funge (2009, 642) propose tweaking the learning rate and using manually set rewards.

4.5. Application

Reinforcement learning is best suited for offline learning. It works well with uncertainty and can be used to solve problems containing lots of different interacting components. Reinforcement learning should not be used if the problem has an easy answer or it has too many states. In addition, it should not be used in situations where strategies change over time. The use of reinforcement learning has produced good results in board games and turn based strategy games. Millington and Funge (2009, 642 - 643).

4.6. Q-learning

Q-learning is a method used in reinforcement learning. The name comes from the set of quality information it holds about states and actions (Millington and Funge 2009, 633). Shi (2011, 378) explains that Q-learning is a model-free algorithm. This means that it does not try to build a model of the world but treats it as a set of states. Millington and Funge (2009, 632) add that model free algorithms are much easier to implement than not model free ones. Russell & Norvig (2010, 831) note that because Q-learning does not have a model it cannot look ahead, which restricts the program's ability to learn.

Q – Learning handles state – action pairs for value calculation. The value for state – action pair is updated based on how good the action was and how good the next state will be. (Equation 1. Shi, 2011)

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \quad (1)$$

where $Q(s_t, a_t)$ is the value of action – state pair

s_t is game state

at is action

α is learning rate

r is reinforcement value,

γ is discount factor

Learning rate controls how much the current value has influence over the stored value. The algorithm does not learn if the learning rate is set to zero. If it is set to one the algorithm does not use stored values. Discount factor controls how much value the following steps have. A high discount factor emphasis good end states and it is only slightly affected by the total path length. A low discount factor also tries to find high end value but it emphasises shorter paths.

5. DECISION TREES

Q-learning works with states and actions. Making an action changes the current state into another one. This means that a decision tree structure can be applied to Q-learning. In a simple system decision trees are a good way of representing states and actions. Bell (2014, 46) notes that decision trees can create complex models when they are used in machine learning. He adds that in order to avoid data over-fitting it is sometimes necessary to prune values into new categories. Figure 3. shows the decision tree structure.

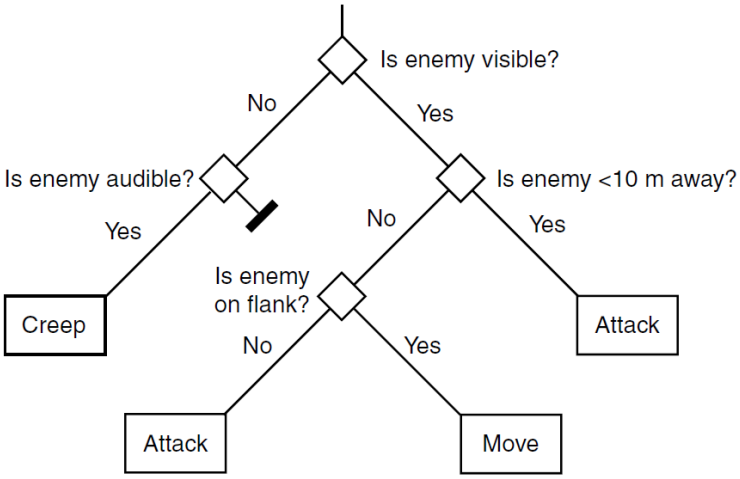

Figure 3. A decision tree (Millington & Funge. 2009, 296)

6. CORRELATION

After the AI has been trained its decision making process is analysed. The terminal states for both losing and winning sequences are used to determine which sub states are the most important for determining the end result. The correlation between sub state values and terminal states is calculated by Equation 2. (Wikipedia, 2015)

$$r = \frac{\sum\left(x-\bar{x}\right)\left(y-\bar{y}\right)}{\sqrt{\sum\left(x-\bar{x}\right)^2 \sum\left(y-\bar{y}\right)^2}} \tag{2}$$

where x is value in array 1

$\bar{x}$ is avarage value in array 1

y is value in array 2

$\bar{y}$ is avarage value in array 2

The equation produces values between -1 and 1. If the value is close to 1 it means that the parameters have a strong positive correlation. Near -1 value means that the parameters have a strong negative correlation. If the value is near 0 there is no significant correlation between the parameters.

7. CODE STRUCTURE

The goal of this bachelor thesis was to make a portable code structure that can be used in multiple projects. In this chapter I have listed all the major classes in my coding structure. The code was done with C# for Unity3D game.

7.1. Sub state

A sub state is a building block for a game state. It is a basic class that only contains information about its identification data and value function. A value function is an integer delegate function that is used to calculate a sub state value based on a given game state. It is worth to note that a sub state value is used only for determining the current game state. It does not have any effect on how good that state is in the reinforcement learning algorithm. Each value function is set to give values between 0 and 10. By limiting the values to small enough range the states can be kept in a reasonable amount. The sub state class does not need to be changed if it is moved to another application.

7.2. State

A state is a collection of sub states. Each sub state increases the total game state accuracy and simultaneously increases the amount of possible states. States may also be set as terminal states which are the only states in this thesis that have rewards. They do not have a value property since Q-learning uses action state pair values. The Rewards are set for the winning and the losing states. During a simulation when one player wins the losing player’s state is given a losing state reward. A state also keeps track on how many times it has been used for simulation data gathering purposes. The state class is not dependent on the application and can be transported to another application without any changes.

7.3. Action

The action class is a collection of delegate functions that determine properties of an action. The delegate functions make the class easily transformable to another application. The delegate functions are set when a new action is created. One of the functions determines whether the action is possible to execute at the given game state. If the action is possible to execute it can be done by using an execute delegate function. Some actions take more than one turn and the action class also has a delegate function for determining if the action is finished. The class also has a continue action function which can be used on unfinished actions. This does not mean that the action must be continued. In case the AI thinks some other action is more important it can choose not to continue the previous one. Finally, the action class contains a delegate function that determines whether the action takes a turn when it is executed. Even if the action does not take a turn to execute it can be only used once per turn. The actions that do not take a turn are given a priority since performing them first means that the AI can make multiple actions before a turn is changed.

7.4. Action State

The action state class contains one action and one state. It has a Q-value parameter that describes how good the action is in that state. An action state is the basic component in reinforcement class calculations. The class also contains temporal data from previous and following action states and it tracks how often a given action in a given state results to a specific state. The previous action states are used when a terminal state has been reached and previous action states are updated. The following action states and following state probabilities are used to calculate an expected value for an action. After the simulation has ended data is recorded to external files and the class is cleared.

7.5. Reinforcement learning

The reinforcement learning class is a base class for the reinforcement handler. It contains methods for getting and adding other class instances such as actions and states. The class also contains a recursive method for updating previous action states. The method is called after a terminal state has been reached.

The main function of the reinforcement learning class is to calculate an optimal action in a given state. The class has two ways of calculating the best action: One for the learning period and the other for a finished product. The learning period emphasizes unused action states in current and following states. It also artificially lowers the expected value of each state. The more times an action state has been used the less value it is given. Reducing the expected value does not prevent the algorithm from using the best path it just braches out more. A finished game does not try to find new paths. Instead it just looks all the possible actions for the current state and chooses the one with the highest expected value.

The reinforcement class collects simulation data for performance evaluation purposes. It tracks used simulation time as well as the number of times the simulation has succeeded and failed. It also tracks how many states have come up and how many of the possible action – state pairs has been used. The class also calculates a correlation between the terminal states and the game results. This data is used for analyzing the importance of the sub states. Finally, the reinforcement class tracks the percentage of each action in every won game. All the actions are included from the first to the last action. The class can be ported to another application without any major changes.

7.6. Reinforcement handler

The reinforcement handler inherits a reinforcement class. It is where all the application specific reinforcement learning methods are collected. The class contains sub state value functions and sub state creation. It is also where all the action related delegate functions are collected. The class also handles data management. All the iteration data used in the reinforcement learning is stored into external xml-files. The file includes the used actions, states and action states with all of their parameters. The stored data is used to evaluate the AI performance.

8. KING OF THULE

The reinforcement learning code is tested on an existing application. It is a turn based strategy game called King of Thule. The game is set in an Iron Age fantasy world. It has an ill king ruling the lands and the death of the king is expected in the near future. Around the king's castle there are between 2 and 6 villages that are controlled either by a human or an AI. Figure 4 shows a blue character in front of a village. Another village, as well as a castle can be seen in the background.

Figure 4. A King of Thule building view.

Each player can have up to three leader characters that can be ordered to do various things. The characters can interact with each other by fighting or trading. The game has three kinds of leader characters: captains, hunters and warriors. The captains are melee fighters and they start from level three. The hunters are ranged units, while the warriors are melee units.

A character's actions are chosen from the available action pool which becomes visible after a character has been chosen (Figure 5). A pop up window appears to give details about each action if a mouse is hovered over them. A player can give only one command to one of his minions during a turn.

Figure 6. The King of Thule character commands.

In addition to the leader characters the game has villagers and warrior minions. Villagers and warriors are spawned in villages. Warrior minions can participate in a battle but villagers cannot. The spawn ratio between villagers and warriors is nine to one. Warrior minions can be taken from a village by a leader character, but they cannot move on their own. Figure 6 shows a battle between two characters. The leader characters are shown on the left side of the pop up and the warrior minions are located next to them on the right side.

Figure 7. A King of Thule battle screen.

Leader characters can battle with other leader characters, villages and castles. A player wins a game if he attacks the castle and wins the battle. An elimination of all enemies also results in a win. A player loses if all his clansmen are killed or if his villages are conquered. If a village loses a battle to an enemy it is going to be under an invasion period which lasts three turns. The new invader acquires the village if the village is not reclaimed or invaded by another player in time. A player can also surrender at any time.

9. GAME SPECIFIC AI DEMANDS

The game AI needs to be divided into a higher and a lower level AI. The lower level AI handles path finding and automated reactions. The higher level AI is used to make a decision on each turn. The higher AI can be further divided into basic reinforcement learning and the game specific AI demands. This chapter addresses reinforcement learning in King of Thule.

9.1. Adversarial solution

The game is adversarial since it is played against either another human or a computer. This means that the adversarial component needs to be addressed in some way. For this game a part of opponent's state is included in the AI state. This means that there is no need to create a separate system to evaluate what the opponent might do. The system learns to how react to an opponent's position the same way it learns how react to the other sub states. When an AI wins it will learn how to take an advantage from the opponent's game state. Furthermore, it is probably more important to learn how not to lose. This is why after each game, the learning is done with both the winning and the losing AI. After an AI losses the current game state reward is set to the minimum value. It is important that there is enough opponent information to deduce why the AI lost.

9.2. Stochastic solution

The game has stochastic elements in it, for example, in the combat system. Stochastic environment is handled by gathering action – state data. The value of each following state is multiplied by the probability of that state given chosen action. Every adjusted state value is summed together to form an expected value. The action that has the highest expected value is the best choice.

9.3. Action possibility

The game has multiple actions that are possible only at a specific game state. Action class has a function that determines if the action is currently possible. Each function that determines whether an action is possible needs to be included in the sub states. This ensures that when an AI evaluates what to do in the current state it will not try to do something that was possible in past state but not in the current one.

9.4. Continuing action

In this game there are actions that take more than one turn to complete. This is why the action class has functions for determining if an action is finished as well as continuing an unfinished action. The AI will continue an action if it is unfinished unless something more urgent appears. For instance an AI may decide to go to forest to pick up berries. It will continue that action unless it sees an opponent approaching its village. A continuing action will not cause problems with reinforcement learning. Since the values of action states are calculated only after the game has ended. After a continuing action is finished it will produce consequences instantly. The game does not have any actions that have delayed effect on the game state after the action or the action sequence has been completed.

9.5. Free actions

Most actions, such as move and attack command, are followed by a turn change. Yet there are some actions that do not change the turn. A player can, for example, vote on a law proposition and move a character before his turn is over. For reinforcement learning purposes these free actions are handled before the normal actions. After a free action is performed the game state is updated before the normal actions are done. This way the reinforcement learning knows which action is responsible for the change in the game state.

9.6. Learning efficiency

The learning happens only after an end state has been reached. The previous action state value is then updated recursively until the first action state has been reached. A value iteration is a non-branching function at first since each action state has only one following state and one previous action state (Figure 5).

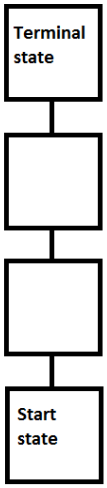

Figure 8. First learning iteration process.

After a terminal state has been reached multiple times, the paths form braches (Figure 6). An action state updates all of the known previous actions states. The updating process works in layers. The first updated action states are the ones that are immediately before the terminal state. After that the previous layer is updated and so on.

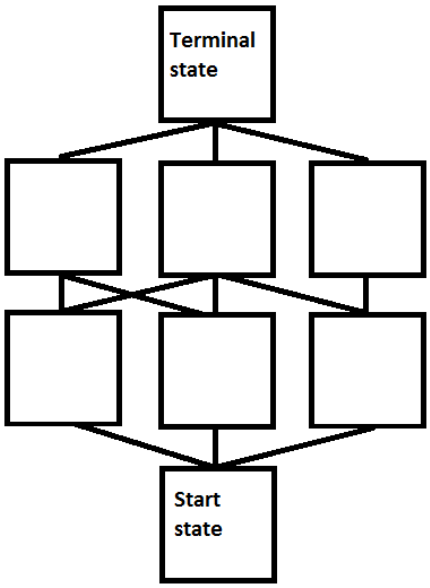

Figure 9. A sample of later stage learning iteration process.

In addition to a learning rate, the reinforcement class uses the probability of an action class to reach a certain state to limit the value change iteration. For instance, if the probability of a following state is 50 percent, the change in the action state value is half from the equation 1. Note that even though an action state updates other action states only once, it may have been updated multiple times by other action states.

9.7. King of Thule actions

The game has several actions that the AI can take. Most actions change turn automatically but not all. The actions that do not change the turn can be performed only once per turn. During an AI's turn the AI chooses the best action or actions then gives the commands and changes turn. All actions that are done with a leader character have separate functions for each character. The AI chooses the one it wants to use based on the learned data. All actions are listed in Table 2.

Table 2. King of Thule actions

Name	Description	Takes a turn
Attack castle	The AI selects one of its characters to attack the castle. The character will ignore other buildings in its way but it will attack enemy ground troops if they get close enough.	yes
Attack village	Attacking an enemy village is done with one of the AI's characters. The target is set to the closest enemy village. The character will ignore other buildings in its way but it will attack enemy ground troops if they get close enough.	yes
Forage	The forage is an action where warrior characters hunt and diplomatic characters collect berries and mushrooms. Foraging gives characters experience points which are automatically used for leveling up. Foraging is not free. Taxes must be paid for the land owner.	yes
Get more units	Getting more units sets the character's destination to the nearest home village that has warrior minions in it.	yes

Heal	Healing is done if the character stays still for a turn and it has been set to heal. Healing affects both the leading character and as well as its minions.	yes
Return to village	Returning to village uses the same function as getting more units except that the leader character will not collect warrior minions. The character enters the village and stays there until a new command is given.	
Run away	Running away is an action where character moves away from a strong close by opponent. Running away is not set to any specific destination. The character just tries to move away in a straight line.	yes

9.8. King of Thule sub states

A state includes a sub state for each action possibilities. This needed to make sure that the AI knows what actions it can do. In addition to action possibilities the sub states contain more generic information about the game. All character related values are separate for each character. All the sub state values are either Boolean or integers with limited range. The generic King of Thule states are listed in Table 3.

Table 3. King of Thule sub states

Name	Description
Battle value	A battle value is calculated from the character's level and the warrior minion amount. There are several sub states that use battle value. The values are calculated for castle, villages and characters for each player.

Diplomacy value	A diplomacy value is an integer value and there are separate values for each player. Diplomacy values are between one and five.
Distance value	Distances values are calculated for characters, villages and castle. The values are normalized to between zero and ten.
Money value	The money value is an integer value between zero and ten. The changes in money amount have larger impact when there is little of it.
Opponents left	The number of opponents is used as a sub state as it is.
Tax value	The tax value is the current total tax rate. It is an integer value between zero and ten. Taxes are paid after successful foraging.

10. RESULTS

The simulation data was recorded first after a training period and again after a play mode simulation. The data was analysed in multiple ways in order to get a good representation of the AI performance. At this point I would like to note than though the reinforcement learning template is done, the King of Thule is not a finished game and the results may reflect that. The general simulation data is shown in Table 4.

Table 4. Reinforcement learning data

Simulation time	1h 7 min 15s
Successful simulations	192
Aborted simulations	0
Average simulation length	11
Total number of states visited	7257
Xml file size	13.64 MB

Table 4. shows that the average game length was elven turns. That seems to be a good number to get a nice game duration. The total number of states is quite large but still manageable. No aborted simulation were necessary which means the simulation never got stuck. The Xml file size shows that the file takes a decent amount of memory. This is due to a relatively large amount of sub states.

The training period had visited 3586 different states. During the last iterations it had used 98 % of all the action possibilities in all of the visited states. After the playing period the total amount of visited states had increased up to 7257 and the average used actions was 59 %. The increases visited state count means the learning process was not complete. In the future some of the sub state values should be approximated more to prevent too large state space. The decreased used average actions shows that in the play mode the AI uses only known states and does not try to find new paths.

The terminal state data was collected and correlation calculation was done for each sub state. The calculation was done with both winning and losing data. The results are show in Table 5.

Table 5. Correlation between terminal states' sub state values and game results

Enemy character battle value	-0.49
Enemy character distance	-0.42
Enemy village battle value	-0.31
Character home distance	0.14
Own village battle value	-0.08

Only the most influential sub states are collected in Table 5. Some of them have a reasonably high absolute value. An enemy character battle value and its distance to player characters correlate negatively with winning. Similarly if an enemy village has a high battle value winning is less likely. As the distance between an owned character and the home village increases the likelihood of winning increases. That is rather simple since the player cannot attack the castle or an enemy village form home. The home battle value does not have significant correlation with winning since it work in two ways. It is better to have as many units as possible with a leader character when it attacks enemies. On the other hand unprotected home village is easier to conquer. It is worth noting that the calculation was done only with the terminal data. The sub states may have different impact to the game in early and middle parts of the game.

In order to analyse action importance all winning sequences were gathered and listed. The used actions are show in Table 6. It shows how often each action was used in the winning sequences during training and play mode.

Table 6. Action percentage in winning sequence.

	Training	Playing
Attack castle	4 %	5 %
Attack village	6 %	6 %
Forage	25 %	25 %
Get more units	14 %	20 %
Heal	3%	3 %
No action	11 %	3 %
Return to village	12 %	15 %
Run away	1 %	1 %
Search map	24 %	22 %

Table 6. shows that the training period and the playing period are quite similar. Though it can be seen than in the playing mode the AI gets units more frequently. In addition in the play mode the AI does not need to resort to no action which is performed only when there are no other alternatives. The search map function is always going to be reasonably high because the AI needs to scout to get information about the map. Similarly after search map has been performed foraging becomes possible very quickly when new lands are discovered. Healing and running away were almost never used which suggest that their availability function or usefulness needs to be improved.

11. REFLECTION

During this thesis I have learned several things about the reinforcement learning application. In this sections I have listed the general things that can be concluded from the data analysis. In addition this chapter handles non data related things that are relevant for future applications.

11.1. Game simulation speed

During the training phase of the game it is important that the game can be played at maximum speed. Reinforcement learning is a slow way of learning because it requires a large number of simulations before it starts to understand the consequences of the actions. The simulation speed was increased by setting Unity time scale to 100.

11.2. Game state initialization

For future projects it would be nice to implement a system that can create any given game state very fast. This way during simulations you could skip going through same game states over and over again. Since the game has limited amount of options in the beginning it does not take too much time go through all possibilities. However as the game goes on the number of possible game states increases fast since each turn is a branching point.

11.3. Performance

The King of Thule's AI can be played against. It works adequately but is worse than human player. There are several things that make the implementation difficult for this game. It is easy for human but difficult for a program to understand is unit positioning and movement paths. Humans can evaluate distances and angles between units and buildings. People can see which unit should be moved and how it should be moved. The game AI does not know how to take a path to destination that will most likely avoid enemies. In addition the King of Thule lower level AI overrides the reinforcement learning commands when character react to objects around them. For instance if opponent moves too close character either attacks or runs away regardless of what it was doing. This adds randomness to the learning process since sometimes given actions are not finished.

11.4. Future application

Since the AI performance was limited by complex action requirements and lower level AI, it would be nice to see how far it can go in a fully discrete game. It should be relatively easy to implement it to either a card game or a board game and the AI should get best results in those.

There are two instances where the game AI chooses actions randomly. They could be changed so that the AI would make more educated guesses. The first and more useful one is when the AI has to evaluate a new state that it has not been in yet but it has data from its neighbouring states. Currently the AI does not do any calculations between existing game states. Even if only one sub state value is changed by only one unit the AI has no idea which action is the best action. Future AI generations could use interpolation to estimate states between known states.

The second situation where educated guesses could be used is when there is no related data to a new game state. Extrapolating outside the known state boundaries does not seem viable. It could be done with a mechanism that would give the AI a preliminary guess of the action state value. This seems unnecessary since the implementation would be game specific and most likely difficult to make. Also after the first usage of the preliminary value it would become useless since the reinforcement learning takes over.

REFERENCES

Barber, D. (2010). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. (http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/090310.pdf) [17 August 2015]

Bell, J. (2014). *Machine Learning: Hands-On for Developers and Technical Professionals*. John Wiley & Sons, Inc., Indianapolis, Indiana.

Coppin, B. (2004). *Artificial Intelligence Illuminated.* Boston: Jones and Bartlett Publishers. (http://coltech.vnu.edu.vn/~sonpb/AI/AIIlluminated.pdf) [17 August 2015]

Daumé, H (2012). *A Course in Machine Learning.*

Ethem, A. (2010). *Introduction to Machine Learning Second Edition.* Massachusetts The MIT Press. (https://www.lri.fr/~xlzhang/KAUST/CS229_slides/chapter18_RL.pdf) [10 Helmikuu 2015]

Millington, I. & Funge, J. (2009). *Artificial Intelligence for games.* Morgan Kaufman Publishers. (http://ldc.usb.ve/~vtheok/cursos/ci5325/lecturas/Artificial%20Intelligence%20for%20Games.pdf) [17 August 2015]

Norvig, P. & Thrun, S. (2015) *Into to Artificial Intelligence.* (https://www.udacity.com/course/cs271) [17 August 2015]

Poole, D. & Mackwort, A. (2010). *Artificial Intelligence: Foundations of Computational Agents.* Cambridge University Press. (http://artint.info/html/ArtInt.html) [17 August 2015]

Russel, S. & Norvig, P. (2010). *v*. Prentice Hall. (http://51lica.com/wp-content/uploads/2012/05/Artificial-Intelligence-A-Modern-Approach-3rd-Edition.pdf) [17 August 2015]

Schwab, B. (2009). AI Game Engine Programming. Cource Tehnology, Boston, USA. (https://mehmetakifsonmez.files.wordpress.com/2013/12/ai-game-engine-programming.pdf) [21 July 2015]

Shi, Z. (2011). *Advanced artificial intelligence*. WSPC: River Edge, NJ, USA.

(2015). *Correlation and dependence.* https://en.wikipedia.org/wiki/Correlation_and_dependence [18 August 2015]