SmarTetris

Xiao Wang, Ling Zhang and Xuliang Qin

CS5100, Northeastern University
December 12, 2016

Abstract

This project implements five different progressive AI agents to play Tetris game. The depth-limited greedy agent uses feature-based search with depth of 2. The reinforcement learning agent use feature-based q-learning to train optimal weights. These two agents are both capable of surviving the game for infinite time.

1 Introduction

1.1 History

Tetris is a famous game invented by Russian mathematician Alexey Pazhitnov. In 1984, Alexey Pajitnov was diligently programming computer games to test the capabilities of new equipment developed by the USSR. In his spare time, he drew inspirations from his favorite puzzle board game, Pentominos. Pajitnov called this game "Tetris," a combination of "tetra" (the Greek word meaning "four") and "tennis" (his favorite sport) [1].

1.2 Rules

Tetris is a complicated game popular among the world. It is played on a 22 * 10 board of block space. Seven set of blocks, called tetriminos, can be used as shown below. At each step, the game will generate a random tetrimino. Tetriminos keep falling from the top to the bottom. Each tetrimino will fill 4 squares on the board. The player can use arrow keys to control the terminal position of each tetrimino. Each line full with block will be removed from the board and produce rewards. Several lines can be cleared at the same time with a higher reward. The goal for player is to clear the lines on the board and survive as much time as possible. The Tetris game is infinite.

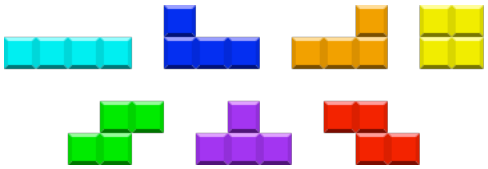

Figure 1: The Standard Tetriminos Set

1.3 Meaning of Tetris AI

In the block space of Tetris. Each block can be empty or filled. However, the trivial representation contains 2^{220} possible configurations of the board, which means it is hard to play well. Our goal is to maximize the lines we cleared, and maximize our surviving time. The offline version of the game, in which the order of subsequent tetriminos is known, has been proven to be NP complete via reduction to the 3-partition problem. In addition, the online version, in which subsequent tetriminos are randomly chosen, is also difficult, because the tetrimino that the computer "plays" is unknown [2].

But the game has a clear game setting. It will benefit our AI approach tremendously. For each action we can get a deterministic game state. Though the steps are unlimited. We have finite number of tetriminos shapes, choices of actions. The score and cleared lines will help us evaluate different game states.

Tetris has an extensive research landscape. Different AI approaches can be used on it with clear rules of Tetris. There're also amount of Tetris AI competitions online.

1.4 Goal and solution

Our goal is to build a SmarTetris AI agent to play Tetris automatically. In order to achieve this goal, we implemented 5 different agents using different AI algorithms including greedy, depth-limited greedy search and feature-based q-learning. Each approach is an improvement of the previous one.

To sum up, we intend to achieve goals as follows:

- Maximize agents' ability to eliminate the number of lines before the end of game
- Design an easy-going framework for different agents
- Make game result repeatability, and record the running video for each agents
- Research appropriate features for advanced search algorithm

2 Background [3]

2.1 Reinforcement Learning

In reinforcement learning, agents use observed rewards to learn an (approximately) optimal policy for an environment by choosing an action to maximize the expected reward giving the current observed state.

2.2 Q-Learning

Q-learning is a model-free reinforcement learning method where we use samples to update Q value as follows:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

Calculate discount for Q value

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(sample)$$

Update Q value

Q values will converge to optimal policy with enough exploration ϵ and low enough learning rate α

2.3 Feature-based Representation

When the state space is too large to represent, it will be impossible to learn all Q we can describe states using a vector of features and a vector w of weights:

$$Q(s,a) = w \cdot f(s,a)$$

Use for getting Q value

And we do the weight update at each time a sample is received:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(error)$$

Update Q value

$$w_i \leftarrow w_i + \alpha(error) f_i(s,a)$$

Update weight

$$error = R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)$$

Get error value

3 Related Work

When designing the agent for Tetris, an important aspect is to invest the mathematical foundations and analyze Tetris features. The goal for a Tetris is to eliminate lines before the end of game.

3.1 Possible features

Features are functions from states to real numbers (often 0/1) that capture important properties of the state [3]. As a result, we list all possible features of Tetris before designing the algorithms, which are shown as follows:

- Existing Holes: holes in current game board (holes means which cannot be reached by instant drop off directly)
- Max Height: the highest column in the current game board
- Total Height: the total height of the sum of all column height in the game board
- Cleared Lines: the lines which have been eliminated
- Delta Height: the sum of difference of each pair of adjunct columns
- Is it the special tetriminos? Such like a square, a straight line or a "z" shape tetriminos?
- Is the heap a Skyscraper? (whether the heap is approaching the celling of game board)

3.2 Mathematical foundations

According to the mathematical proof provided by [Brzustowski, 1992, Burgiel, July 1997] in *Can you win at tetris?* [4], with a sequence of tetriminos, the eventual termination is guaranteed to enter an end in a well of width 2(2n+1), with n being any rumination.

When using a winning strategy, we will find that the same well states will appear over and over. This is because

	Well Width=2		Well Width=2n, n>1		Well Width=2n+3, n>=0	
Piece	Height	No. of States	Height	No. of States	Height	No. of States
Square	0	1	2	n	(no winning strategy)	
Kink	2	2	4	2n	(no winning strategy)	
Bar	4	2	4	2n	4	2n+3
Elbow	2	2	5	2n	8	4n+6
Tee	2	3	4	3n	7	5n+5

Figure 3.1 A summary of winning single-piece strategy

there is only a finite number of well states with no full cells above row 22. Since each of the 10*22 = 220 cells in or below row 22 is either full or empty, there can be at most 2 * 2 * … * 2 =2^{220} different states in a winning strategy. [4]

The winning strategies for single pieces are summarized in Figure 3.1 A summary of winning single-piece strategy. [4], which lead to the gradual accumulation of persistent blocks and eventually the termination of the game.

Based on the winning strategy analysis of [Brzustowski, 1992, Burgiel, July 1997], we can get some useful features from the possible features which we guessed at the beginning.

4 Analysis of Tetris NP-Hard Work

In paper "Tetris is Hard, Made it easy" [5], The Tetris has been proven to be a NP-complete problem.

As mentioned in [5], Tetris is a 3-PARTITION problem. For Tetris, three Partitions are R, W, H:

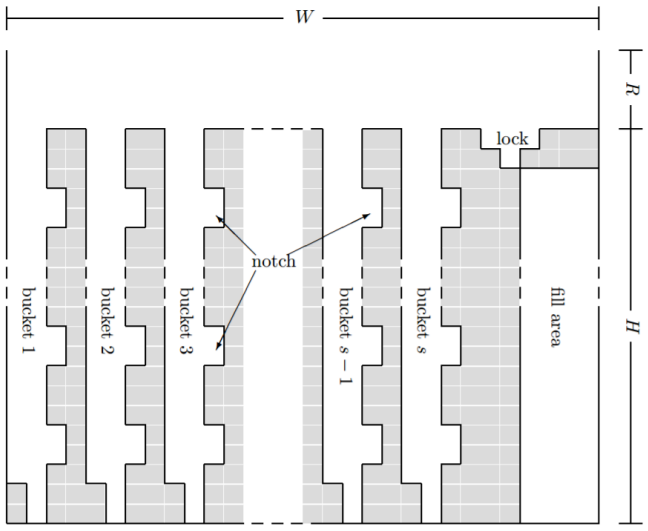

Figure 4: Initial Tetris Game Board [5]

-

-

- R is the space needed to rotate the translate the pieces. In Standard Tetriminos Set which is showed in figure 1, we have seven different tetriminos.
- W is the width of game board and I equal to 4s+6
- H is the height of the bottom part of the game board that needs to be cleared and is equal to 5T + 18

By defining a smaller initial game board, a smaller and less complex sequence of Tetris pieces, the Demaine, Hohenberger and Liben-Nowell [6] and Ron Breukelaar, Hendrik Jan Hoogeboom and Walter [5]. Using the reduction proposed a "no" and "yes" instance of 3-Partition reduces to an instance of Tetris of which the game board cannot be cleared [5].

In one word, as a NP-Hard problem, the Tetris limits us to solve this problem by using search algorithm such like Breadth First Search, Depth First Search and Uniform Cost Search. When using these search algorithm, the time complexity and space complexity is too large to run in limited time. We will also analyze similar situation in different agents which is designed to solve Tetris problem.

5 System Architecture

The states we use in our algorithms should be able to represent the game states completely and distinctively. Thus, our states are designed as (board, score, block, next Block):

- Board: a 2-d list of integer, where 0 represents empty space and others represents different blocks
- Score: an integer of the total score of the current game
- block: a 2-d list of integer representing the current given block, placed at top center without rotation
- next Block is like block but will be given at the next move

5.1 Actions

The real actions in game can only move blocks with one square. But using the real game actions will make our pro

grams more time-consuming. Thus we simplify the actions used in our programs as (rotateN, x):

- rotateN: an integer in {0, 1, 2, 3}, representing the number of times to rotate the block clockwise
- x: an integer in [0...MAX_NUM_COLS], representing the x-position to place the block

After rotating and placing the block, we hard drop it. This is a simplification because real game allows further horizontal move after drop. But this is a reasonable simplification because it makes generating next states possible and still can solve current state.

5.2 Data Structures

We implement our state as a class, which has a method to generate possible successor states. We implement different algorithms as subclasses of state, which can generate an optimal move under current state.

6 Implemented Agents

We implemented 5 different agents:
Random Agent, Minimum Height Agent, Greedy Agent, Depth-limited Greedy Agent and Q-learning Agent.

6.1 Baseline – with Random Agent

The baseline aims to get the minimum performance for the Tetris game. Our baseline is designed to get random agent action for the game, described as follows:

Given: An initial game board (which is 22 in height and 10 in width) and a finite sequence of Tetris pieces.

Question: When meeting with new tetriminos, what is the performance of Tetris to deal with random tetriminos and action? And also the translation and rotation is also random. What will the average score for Random Agent policy?

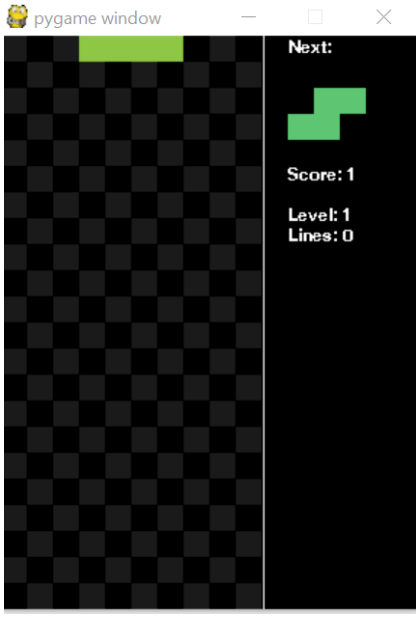

Figure 6.1 Initial Game Board

As is discussed in part 4, the Tetris is a NP Hard problem. With random action, the Tetris get very limited performance.

The Evaluation of Baseline

After running ten times, the average score is showed as follows:

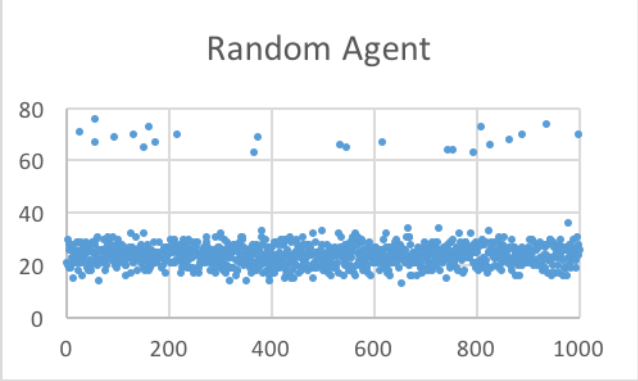

Figure 6.2 Random Agent Score Distribution

Average score: 24.756

Normalize variance: 0.09662975.

Compared with our goals, maximizing the number of lines cleared before the end of game, it is obvious that the baseline does not satisfy the requirement.

As a result, the Tetris approaches to the end in very short time.

6.2 Minimum Height Agent

The Minimum Height Agent aims to put the tetriminos at the current lowest position in the game board.

Given: An initial game board (which is 22 in height and 10 in width) and a finite sequence of Tetris tetriminos.

Question: When meeting with new tetriminos, what is the performance of Tetris to put the newest given tetriminos at the lowest position? What about considering dropping directly or considering all possibilities?

The Minimum Height Agent aims to put the tetriminos at the current lowest position in the game board. It is a better way to maximize the performance of Tetris than the random agent. The pictures in figure shows how we put tetriminos at the lowest position.

In our algorithm, at first, we search each row form the top to the end of the game board. And at each row, searching the row from left to right and check whether there is a pixels or not.

As a result, in the initially game board, the algorithm will put the first tetriminos at the left bottom. And for the second tetrimino, it will put in the middle until the first line is full, which is shown as follows:

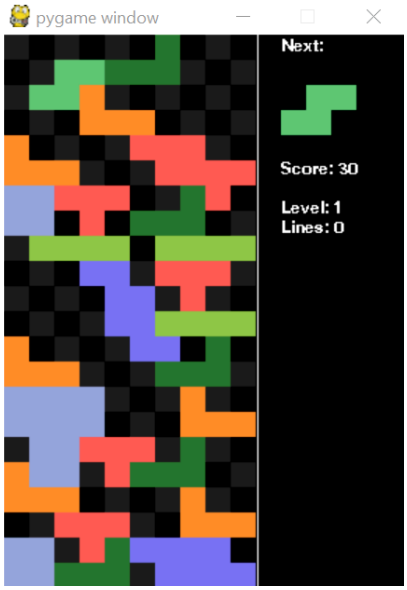

Figure 6.3 The Normal state of Minimum Height

As is shown in the *Figure 6.3 The Normal state of Minimum Height,* we will put each tetriminos at the lowest height position. Because we do not consider moving the tetriminos when it arrive at the end of game board. It may cause the situation that next tetrimino put directly on the wrong position.

The Analysis of Space Complexity

It is obvious that considering move tetriminos when reaching the bottom of board will get an ideal result. Why we do not code in that way?

To get minimum height, we need check the conflict of the given tetriminos with current board after each move. When we consider all situation of each tetrimino, the time complexity and the space complexity is too large to get a result in limited time.

When we just drop the tetriminos in a straight line, and just consider one rotation, the time complexity and space complexity are both $O(n^2)$.

When we consider every rotation at each position, the time complexity and the space complexity will be greater than $O(2^{220})$, which is not acceptable for our program.

For instance, the domain of tetriminos has seven, which is shown in the figure 1, and each tetrimino has at least 2 rotations. For the game board, we have 22*10 position.

The Evaluation of baseline

After running ten times, the average score is showed as follows:

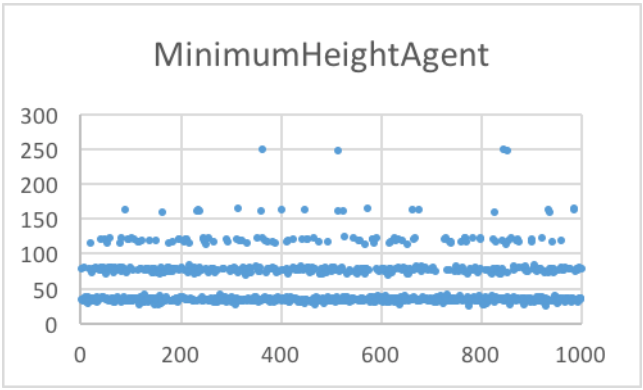

Figure 6.4 Minimum Height Agent Score Distribution

Average score: 58.705

Normalize variance: 0.334299294.

Compared with baseline – Random Agent, Minimum Height has a much better performance. While when aims to maximize the number of lines cleared before the end of game, it is obvious that the minimum height agent distribution does not satisfy the requirement.

As a result, the Tetris approaches to the end in very short time.

6.3 Greedy Agent

The Greedy Agent aims to execute the best move according to the best value in the next step. We use a linear function to calculate the value for each state.

Question:

1. What move should be treated as the best move?
2. What are the most important features that compute the value of a state?
3. What is the weight for each feature?

What is the best Move?

- Best move tries to clear lines at first thought (more lines at the same time is better).
- If can't clear lines, best move should avoid filling block above an empty square.
- Best moves will keep a low fluctuation, avoid possibility of hill.

To satisfy these criteria, we implement the following features.

The Selection of Feature

Max Height:

Height of a column is the highest filled square in each column. Basically, a lower max height indicates an easier game state. So we take the max height of a state as one of the features.

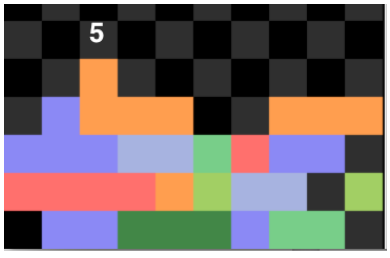

Figure 6.5 Max Height = 5

Total Height:

Only max height is limited on judging a state. We also calculate the total heights. Total height is the sum of the heights of each column.

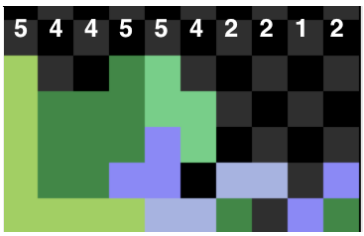

Figure 6.6 Total Height = 34

In this case, by sum (5,4,4,5,5,4,2,2,1,2), we got the total height 34

Holes:

A hole is an empty square with filled squares above it. Having holes is a bad characteristic for a state because holes are hard to clear. We need to clear all the lines above before fixing the holes.

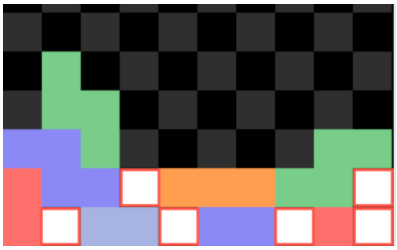

Figure 6.7 Holes = 6

Lines Cleared:

In the rule of Tetris, if we clear a line we will get 40 score. If we clear more than one line at the same time we will get 100, 300, 1200 for clearing 2, 3, 4 lines. With more lines cleared we will get a more advantage state. That's why we calculate lines cleared.

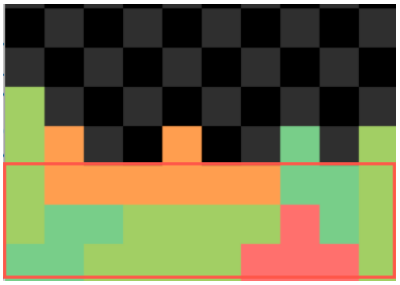

Figure 6.8 Lines Cleared = 3

In this case, three lines is going to be cleared at the same time. So the lines cleared is 3, and will add 300 to the score

Delta:

During the test we found the situation that tetriminos keep falling in one column. Only long strip can save this situation. So we add Delta into feature. Delta is the sum of absolute delta value between two adjacent columns' heights. We already given the definition of height in the Max Height

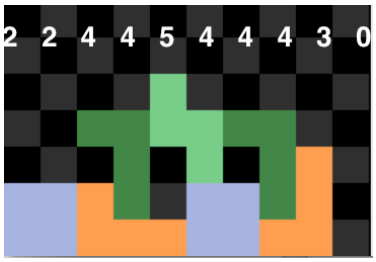

Figure 6.9 Delta = 8

In this case, calculating the delta using following formula:

|2-2|+|2-4|+|4-4|+|4-5|+|5-4|+|4-4|+|4-4|+|4-3|+|3-0| = 8

With lower delta, the agent will try to place tetriminos in lower height columns.

The adjustment of weights

At this step, we haven't implemented reinforcement learning. So, we try to do assign these weights manually.

1. Max Height: we want a lower max height to get a more advantaged situation, and it's not a most significant feature. So we assign it -1 at first.
2. Total Height: we want a lower height to guarantee agent try to place tetriminos in lower squares. Assign -1 at beginning.
3. Holes: the hole is a high priority feature, and fewer is better. Initially, assign -2
4. Lines Cleared: this feature effected most on the score, and have a positive correlation. Assign 2 at beginning.
5. Delta: delta has a negative correlation with score. Low delta can control the fluctuation of the board. So we assign -1 at beginning.

The Analysis of Space Complexity

For current greedy agent, we just consider one step in advance. As is shown in the *Figure 1: The Standard Tetriminos Set [4],* we have seven different tetriminos, and the according to the part 4, considering R, W and H. And these data are relevant with the game board length and height.

In program, each tetriminos has 4 different rotations, and we have 10 columns according to board game width. As a result, the time complexity and space complexity are both $O(n^2)$.

When run greedy agent, we also need check the conflict of the given tetriminos with current board after each move, like Minimum Height Agent. With the same reason, we do not consider all situation of each tetriminos.

Because the time complexity and the space complexity is too large to get a result in limited time.

When get find the best action with the best next step score, we will drop the tetriminos directly to the best action position. With a not good enough weight of Tetris, the performance can just run games around five seconds. It is not an ideal algorithm to solve Tetris problem.

The Evaluation of greedy agent

After running 1000 times, the data is showed as follows:

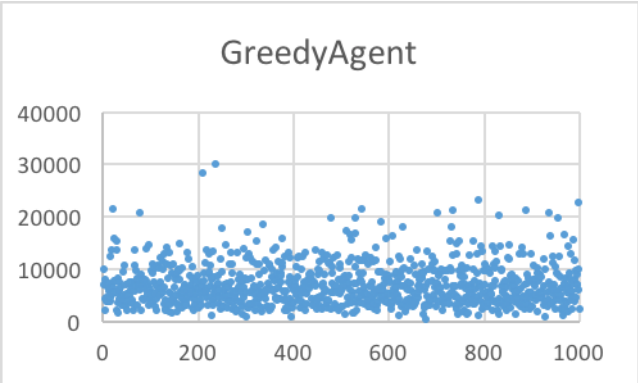

Figure 6.10 Greedy Agent Distribution

Average score: 6883.101

Normalize variance 0.317827668

Compared with baseline – Random Agent and Minimum Height, Greedy Agent has improved a lot. While when it aims to maximize the number of lines cleared before the end of game, it is obvious that the greedy agent still will end game in a short time.

As a result, Greedy agent is still not a satisfying solution for Tetris problem.

6.4 Depth-limited Search Agent

In order to improve the performance of Greedy Agent, we decide to take advantage of the information of the sequence of future Tetris pieces after the current one.

Given: An initial game board (which is 22 in height and 10 in width) and a finite sequence of Tetris pieces.

Question:

1. How to determine the best move given the sequence of future Tetris pieces?

2. How many steps can we look ahead?

3. How does this agent perform compared to Greedy Agent?

How to determine the best move given the sequence of future Tetris pieces?

- Keep using the feature evaluation method de-scribed in Greedy Agent

- List all the possible states after placing current piece and N future pieces.
- Evaluate all the possible states with the same features as Greedy Agent
- Take the action leading to the state with the best evaluation score

How many steps can we look ahead?

As mentioned in System Architecture, action is a (rotateN, x) tuple, so there are 4*10=40 different actions for each state. Thus, for a search with depth N, we have time complexity of O(K*40^N). K is the time complexity of evaluating the states, which is approximately 220. In order to take actions efficiently, we should be able to get one action in 1s, therefore N should be no more than 3.

How does this agent perform compared to Greedy Agent?

We implemented a depth-limited search agent with depth 2. With the same state evaluation features and weights, our depth-2-limited search agent can play the game for infinite time.

6.5 Q-Learning Agent

In previous algorithms, we use manually assigned weights to calculate the score from the features of a state. In order to learn the optimal weights for playing, we take the next step to use Q-learning to train the weights of selected features.

$$Q(s,a) \leftarrow w \cdot f(s,a)$$

Use for getting Q value

$$w_i \leftarrow w_i + \alpha(error) f_i(s,a)$$

Update weight based on error action

$$error = R(s,a,s') + \gamma \max Q(s',a') - Q(s,a)$$

Get error value

In our training, R (s, a, s') is the number of lines cleared from s to s', Q (s, a) is the Q-value calculated using a linear function within features at next state.

6.5.1 Initial Training

We use the same features and manually adjusted weights into Q learning. While, the result is disappointing. The learning result of Q Learning agent cannot perform good enough to solve Tetris problem. Some tetriminos even placed as a tower without lines cleaned. Which made us consider where the real problem is?

Reviewing the update procedure, we found the weights are updated too quickly. After 3 episodes some weights already reached 1.0E+10. Because features like total heights may be a huge number, and will get increased dramatically. Once a feature has a too large figure, it will take a more essential role on the next evaluation.

6.5.2 Improve Feature

Reconsidering all the features and changing features:

- Max Height -> Delta Max Height
- Delta -> Difference between two states' delta
- Holes -> Delta Holes
- Get rid of the total height

	LinesCleared	TotalHeight	MaxHeight	Holes	Deltas
Initial	4	-2	-1	-1	-1
episode 1	3.881490037	5.149721621	6.587757221	-35.66024158	0.635713852
episode 5	1.05E+59	1.21E+120	-1.30E+121	4.38E+121	-5.46E+122

Figure 6.11 Q Learning Process

7 Analysis of Data

In order to compare the efficiency of different algorithms, we collect data and analyze the statistics.

Normalized variance: we first divide each data by their average, and calculate their variance.

Our Expectation:

1. Baseline: Very stable, and can't reach 40 score (clear one line)
2. Minimum height agent: Very stable, and hardly can reach 100
3. Greedy agent: has large fluctuation, may have some cases with high score.
4. Depth limited greedy agent: Really astonishing result, the agent can almost survive for infinite time with appropriate weights.
5. Q learning agent: optimize our weights, also satisfy infinite game play.

Analysis:

For baseline, the agent places tetriminos randomly, which is almost impossible to clear a line. That also explains why this algorithm gets a steady outcome.

When evaluating the minimum height agent, we can see a great progress. But a high variance indicates that luck still play an important role in this algorithm. From the diagram, we can see the data have a score range from 20 to 300. That prompts us to look for a better algorithm.

For greedy agent, both average and variance take a leap. In this algorithm, the agent begins to think according to the weight of each feature. The max score for this algorithm is around 20000 to 30000. This agent can already deal with most situations, but can't handle some special cases.

After finding the best choice in one more step, the output is satisfying. The agent can survive for long enough.

That's the main reason why we didn't run for a larger number of times, and show diagram as the other algorithm.

During our tests, depth limited greedy agent will finally dead if we run for long enough.

The result after training is nearly an infinite game. During our record, more than 10000 lines are cleared. That is the greatest progress we've ever made. We can conclude from the outcome that optimized weights can increase the surviving time in a great extent.

Average Score	Random Agent	Minimum Height Agent	Greedy Agent
10	24	51.6	7143.3
100	25.93	56.92	7473.21
1000	24.756	58.705	6883.101

Figure 7.1 The Average Score for Agents
Note: The Depth-Limited Greedy Agent and Q-Learn Agent can run game infinitely, so don't have data

Normalize Variance	Random Agent	Minimum Height Agent	Greedy Agent
10	0.025347222	0.163316507	0.108576377
100	0.142430249	0.463788158	0.336619846
1000	0.09662975	0.334299294	0.317827668

Figure 7.1 The Normalized Variance for Agents
Note: The Depth-Limited Greedy Agent and Q-Learn Agent can run game infinitely, so don't have data

Actual Outcome:

1. Baseline: the same as expectation. Low average score, but steady.
2. Minimum height agent: lower than expectation with a high variance. Some special case can clear more than 5 lines.
3. Greedy agent: out of expectation, with an acceptable normalized variance. We even got a 30000 sample which surprised us.
4. Depth limited greedy agent: better performance than greedy agent, usually steady.
5. Q learning agent: better performance than greedy search, with a stable variance.

According to all the data and video result, we basically achieved the expecting developing model. Starting from the baseline, and keep evolve the version. We can clearly get from the diagram that the agent is truly getting better and better.

8 Conclusion

Our project implements and compares different Artificial Intelligence agents for Tetris:

- Random Agent
- Minimum Height Agent
- Depth-Limited Greedy Agent
- Q-Learning Agent

Based on our analysis about these different agents on playing Tetris. Although Depth-Limited Search Agent has some shortcoming, it still performs well to get a high score. It can nearly play the game in a very long time, which you can refer to our demo named

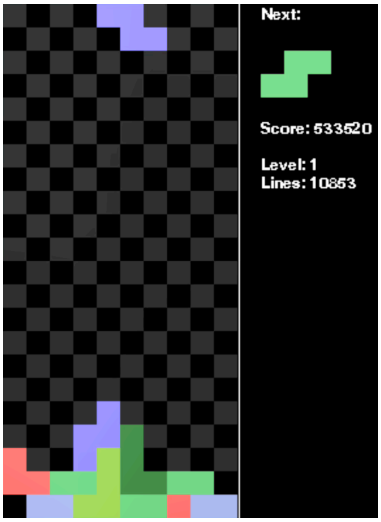

Figure 8.1 Best Score for Deep-Limited Search Agent

"Depth-Limited Search Demo". By discussing why its performance better than Q-Learning Agent, we find for Tetris game, its specialty causes this result.

For Tetris, when you deal with current tetriminos, the next tetriminos is already known. The Depth-Limited Greedy Agent can arrange the best action according to the given data.

As a result, just when the depth of Depth-Limited Greedy is two, its performance is good enough to solve the Tetris. Our best score for Depth-Limited Greedy Agent is eliminating over ten thousand lines, and still keep running without and obstacle.

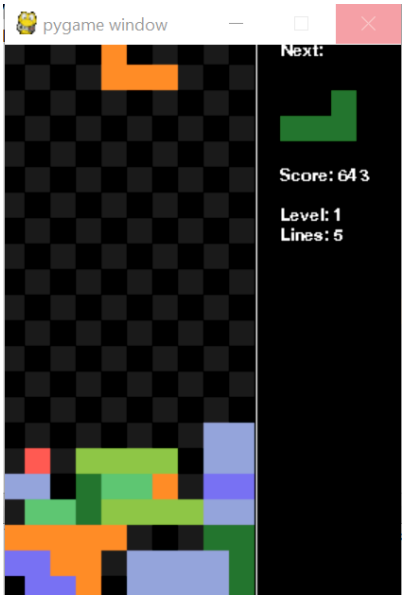

Figure 8.2 Bad Choice for Q-Learning Agent at beginning

When it turns to another acceptable agent, it is Q-Learning Agent. After enough training, Q-Learning Agent can also run ideal times and get a preferable score. While at the beginning, the Q-Learning may get a bad choice because of not enough data, it will greatly affect the following action. At most of time, the Q-Learning failed because the bad performance at the beginning.

The Q-Learning also has a good performance and can run very long time. What's more, the performance of Greedy Agent is largely based on the given weight of features.

Another Minimum Height Agent is something like a simple Reflex agent, each time, it will find the lowest height in the current board. And then put the tetriminos directly to this position. As a result, its performance is a little better than baseline—Random Agent.

In one word, our Depth-Limited Greedy Agent, which looks forward one step in advance, and Q-Learning Agent, which is after enough training, can provide a good enough performance in Tetris game.

9 Discussion

During our project, we applied what we learned in CS 5100, Foundations of Artificial Intelligence in our algorithms, such like MDP, Reinforcement Learning and Probability. The aim of this project is to build an AI agent for Tetris game using search, reinforcement learning and genetic algorithms. The automatic agent is expected to play better than humans in acceptable time. This project also tries to extend the original rules of Tetris and solve the more challenging problem. And also, we greatly improve our programming skills and learn how to do great team work, which is described in each part.

Planed work:

1. Analyze the problem, research the rules in detail
2. Data design: states, features, actions, rewards, discount
3. Infrastructure: build a game engine to generate blocks, calculate scores, count time and visualize
4. Try and implement different algorithms: Search (challenges: time complexity)
5. Reinforcement learning (challenges: offline issue, training time)
6. Genetic algorithm (challenges: local optima problem, randomness)
7. Evaluate the algorithms based on scores, survival time and efficiency
8. Compare the algorithms, improve them if possible

In our project, we finished most of our goals, such like building a game engine to generate blocks, calculate scores and some different algorithms. While for Genetic algorithm, we have got some idea and did some related achievement, the genetic agent is not as good as the greedy agent and reinforcement learning agent. As a result, we will keep doing relevant work on it, and has a more comprehensive understand about genetic heuristic.

10 Future Work

In our project, we do not change the level for Tetris game. With higher level, there will be a higher speed to drop tetrimino. At the beginning, the tetrimino will drop one block at each 1000 millisecond. In the highest level, it will move one block each 100 Millisecond. Except our Depth-Limited Greedy Agent, other four agents can deal with the highest level. And the Depth-Limited Greedy Agent need more advanced Central Process Unity to run in acceptable time. We need improve some inappropriate designs in the framework. But it need huge work.

Besides agents talked about in out paper, we also do a lot work on designing Genetic Agent for Tetris. For many such applications it is likely that some test cases optimally require more computation cycles than others. Therefore, programs must dynamically allocate cycles among test cases in order to use computation time efficiently. [7] With current feature, Genetic Agent will be more effective than our Depth-Limited Greedy Agent. What's more above all, it can greatly reduce the running time, which is very important attribute for long time running.

Interesting is the best motivation. Watching Tetris AI agent makes us feel accomplished, while it is a better choice to design a GUI to see two agents fighting with each other. What's more, in our current algorithm, we just consider the Instant drop for the tetriminos. Can we add consideration about Soft Drop? Will this cause a perfect Artificial intelligence agent for the AI. It is really attractive topic.

To sum up, we will do following work to improve our project Tetris:

- Improve our framework and modify inappropriate fundamental structure
- Complete the Genetic Agent for Tetris
- Find the best features for Tetris Agent
- Consider Soft Drop for Tetris Agent

11 Appendix

11.1 Definition and Terminology

- Board: The Game Board of Tetris. In our project, it is a 22 row * 10 col matrix. When a new tetriminos collide with current board, the game is over.

- Block/Piece: A tetriminos. This is the piece generated randomly and used by us to play. Each block played worth 1 point.

- Cleared Line: A row on game board with all 10 squares filled with tetriminos. This kind of lines will be removed as soon as they appear. The lines above them will drop and fill in the space. Each cleared line worth some points based on how many are cleared at the same time.

- Soft Drop: After a new piece drop on the top of existing pieces, it's allowed to move the piece horizontally for several steps given no obstacles. This rule is not used in our project in order to simplify the state transitions.

- Hard Drop: After a new piece drop on the top of existing pieces, it's not allowed to move the piece any more. This is the rule we use.

11.2 Demos: Videos included in project folder

Baseline Agent: Random.mov
Minimum Height Agent: Minimum Height.mov
Greedy Agent: Greedy.mov
Depth-Limited Greedy Agent: 2greedy.mp4

12 References

[1] http://tetris.com/about-tetris/ The official website of Tetris

Saagar Deshpande, Louis Li, Brandon Sim TetrAIs

[2] Search Based Artificial Intelligence Program for Autonomous Tetris Gameplay, Computer Science 182, Harvard University 2013

[3] http://inst.eecs.berkeley.edu/~cs188/pacman/home.html

[4] John Brzustowski. Can you win at tetris? Master's thesis, University of British, Columbia, 1992.

[5] Ron Breukelaar, Hendrik Jan Hoogeboom, and Walter A. Kosters. Tetris is Hard, Made Easy, Liden Institute of Ad-vanced Computer Science, Universiteit Leiden, 2002

[6] E.D. Demaine, S. Hohenberger and D. Liben-Nowell, Tetris is Hard, Even to Approximate, Technical Report MIT-LCS-TR-865 (to appear in COCOON 2003), Laboratory of Computer Science, Massachusetts Institute of Technology, 2002.

[7] Siegel, E., and A. D. Chaffee (1996). Genetically Optimizing the Speed of Programs Evolved to Play Tetris. Advances in Genetic Programming, Volume 2.