Acknowledgement
The original Tetris game in python was implemented by the user silvasur on GitHub. We offer many thanks to the original implementation of Tetris game. The original version of the game is from: https://gist.github.com/silvasur/565419. Also, the original version is in python 2 and our projected is implemented in python 3.

Software Running Instruction
First, there are two version of the game. One is implemented with learning part and it would play and learn the game with zero knowledge at the beginning. Another is implemented with our optimised results and evaluation function after the learning process. This version would always play the same strategy. Before running the files, please make sure the corresponding modules are installed: pygame, scipy and numpy. If any additional package is required, please install it also. To run the program, you can use the command prompt in windows or terminal in Mac or Linux and run the files with this command: “python [filename].py” or “python3 [filename].py”.

Introduction
Tetris is a tile-matching puzzle video game and it is one of the most famous game in the world. It is played on a board with 20 rows and 10 columns. There are seven shapes of Tetriminos, and Tetriminos fall from top of the board and come to rest at the bottom every time. The player has to control the Tetriminos by moving left and right and rotating clockwise, and the player is needed to get the score as high as possible by clearing lines, which is filling the horizontal row in the board.

The rules of Tetris are very simple but it is not easy to master it, like clearing more 500 of lines. Players need to find the solution to keep the board in good shape meanwhile clear the lines as many as possible. More lines have been cleared, the falling speed of the Tetriminos increases, which means the difficulty increases and the player has less time to decide the best move in the later part of the game. Therefore, the player will be prone to make the wrong decision, and finally lose the game.
In this project, we aimed to implement an AI playing Tetris using and feature-based representation and reinforcement learning. This AI can be divided into two parts, the decision-making part for optimal action and learning part for the evaluation functions. When the Tetrmininos is generated in the top of the board, the AI will search the optimal action every time using the evaluation function based on the state of the current board. And its goal is get the score as high as possible and to “survive” in the game for long period. After losing the game, it would adjust the current and optimal evaluation function and the performance in the game.

**Problem Analysis and Implementation**

1. **State**

   The number of states of a Tetris board is a large if we record whether the stones exist in each box. The number of possibilities is about $2^{209}$ thus it is impossible to save the graph of the board in the database to decide the next optimal move of the AI. To avoid the problem of memory usage, we represent the states of the game by different features of the board:

   i. **Total sum of the peak of each column**: Heuristically, this feature of the board should be minimised for preventing losing due to the congested board.

   ii. **Emptiness of the board**: This feature count how many empty blocks under a piece of filled block (stone). In other words, it is the total sum of the peak minus the total sum of the filled block. This feature should also be minimised as the emptiness would hinder the clearance of rows of blocks.

   iii. **Total number of cleared lines**: Obviously and heuristically, this feature can represent how well players perform in the game and it should be maximised throughout the game.

   iv. **Surface roughness**: This feature quantifies the roughness of the top surface of the blocks. Heuristically, smoother the top surface of the blocks is, easier to clear rows of stone it is. Also, this feature should be minimised for easing the filling of gaps in the game.

   With these features, we can generalize the states of Tetris to linear combination of variables, which is easily to be learned for AI.

2. **Action**

   For every piece of stone, the choices of action are the final locations of the stone after dropping. We group the action rather than replicate the keys played by human. Therefore, we formalized the actions by representing the direction and number of times it moves and rotation of the
Tetriminos, finally instantly drop the Tetriminos without waiting. We implemented the action in our problem like this:

```python
for i in range (0, rotate):
    self.action_list.append(rotate_index)

for i in range (0, abs(movement)):
    if movement > 0:
        self.action_list.append(right_index)
    if movement < 0:
        self.action_list.append(left_index)
self.action_list.append(insta_drop_index)
```

3. Evaluation function

\[
\text{Value of the board} = \sum \text{weighting}_i \times \text{feature}_i
\]

Using the feature previously mentioned, we used a linear function to represent the value of the board, with the weightings for each feature. For the reason of exploration, the weightings are randomized at the early stage. And in the later part of the game, the weightings follow the optimised evaluation function in the reinforcement learning part of the AI.

4. Optimal action

To determine each optimal movement of the Tetriminos, we search the value of the board using the evaluation function for every final possible location for each new dropping stone. The algorithm is in the following:

```python
for num_of_rotate in 0 to 3:
    for movement in range(max_move_left_amount, max_move_right_amount):
        value=move[num_of_rotate/movement];
        if max_value<value:
            max_value=value
            best_move=[num_of_rotate/movement]
```

The AI will calculate the value of the board after placing the stone in every possible location using the evaluation function above. And it acts the most optimal action with the most optimal board value. As the next Tetriminos is given to the player, AI can optimize the move by searching the value of the current board for current stone and next stone. However, it is time consuming for AI to search all the results due to the large number (>900) of possibilities. It leads to the situation that the AI will spend time to calculate the optimal move while it is waiting in the game, which is not efficient. Therefore, we decide to extract three best moves for current stone and searching their maximum value based on the next stone to decide the optimal
movement. It saves a large part of searching time because most of the decision is not optimal and we ignore them in the consideration. Hence, it can improve the searching speed of the AI effectively.

![Diagram showing AI determining optimal move](image)

This figure showed how the AI determine the optimal move

**Reinforcement Learning**

At the beginning, we will explore the weighting of the features by randomizes their weighting. After playing a certain amount of games, the exploration rate decreases based on an exponential decay curve, and the decision of movements would depend more on the optimised evaluation function rather than a randomised evaluation function.

In the learning process, the AI stores the weightings of all attempts as well as the corresponding score in the game. The optimised weighting for each feature is the weighted average of attempted weight. This means:

\[
weighting_i = \left( \sum weightings_j x score_j \right) / \sum score_j \quad \text{for each feature } i
\]

In our calculation, the weightings are multiplied by the square of the score, rather than score only. We suggest that higher the scores, more influences by the corresponding weightings should be. A linear relationship is not enough to magnify the influence of higher scores. If a game with very high score played, the weighting would be changed significantly. Lastly, the all the weightings would be normalised and -1 ≤ weighting ≤ 1.

Since the starting stage of this learning process heavily depends on the randomisation, the resulting weightings from the learning process may varies from game to game. However, they are all optimised to achieve the highest score as they can, and this can be regarded as different playing styles.
To show the process of reinforcement learning, three files would be outputted, and records would be written before and after each game. Those three files are for storing the score (score.txt), the optimal off-line weightings (weighting.txt) and the current on-line weightings (explore.txt) for the evaluation function. In each file, the first column is the game number. In the score.txt, the second column is the final score of each game. In the weighting.txt, the second to the last columns are the optimised weightings for each feature, and it is updated after each game. In the explore.txt, the second to the last columns are the current weightings used in the games, and it is updated before each game.

![Pygame window image]

The picture is an example that the AI played at the beginning stage. This shows that the AI has no knowledge about playing Tetris.

Results

The resulting weighting for each feature after 345 games:

<table>
<thead>
<tr>
<th>Features</th>
<th>Sum of Peaks</th>
<th>Emptiness</th>
<th>Cleared Lines</th>
<th>Surface Roughness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weightings</td>
<td>-0.00723970259470</td>
<td>-1.0</td>
<td>0.01157860299340</td>
<td>-0.20880163779281</td>
</tr>
</tbody>
</table>

The average score of those 345 games is 1803025. The highest score is 39043681. The higher score AI can get if it trains more times of game. After training for about 345 games, we
have got the current model. If it follows the optimal policy we have given to it, the AI can get over 1 million score in one game at most of time. The highest score recorded is much higher than most of the human player. This AI is a high-level player of the Tetris game.

^The Tetris AI plays well after exploration process. The score of that game is over 5,000,000 and the more than 900 lines of block cleared.

**Limitations**

Since the game is only represented by four features, it may not be able to capture some special cases. One or two more features may be introduced to have a better and accurate representation of the game. For example, if we take how sum of stones above the empty in one of our features, then the AI would no longer to put too many Tetriminos above the empty, therefore increase the efficiency to clean the empty. Also, based on our observation, some strategies played by the AI may heavily rely on some particular pieces of stones. If those stones do not appear for a long period of time, the board would become unfavourable for the AI to play and therefore increase the risk to lose the game.
Conclusion
Our Tetris AI is nearly perfect as our algorithm is well designed and trained. Our reinforcement learning is very useful and effective to play Tetris game as Tetris AI does not require to learn from lots of game. We can simplify the state of the board in four features and determine the value of the board using the optimal weighting among these features. We can easily get the final weighting of the features based on the score and get the optimal weighting to make the AI perform well in the game. In conclusion, we have applied what we learned in this course, and we are able to apply this techniques in AI field in this project.

Appendix
In the folder “Appendix”, files “explore.txt”, “weighting.txt”, “score.txt” are the sample data for reinforcement learning; “ai_tetris_with_learning.py”, “ai_tetris_optimised_without_learning.py” are the implementation of the AI Tetris. Pictures and Video clips are also attached in the folder. The picture and video clips named with “Learning” are the sample of the version with learning and the those with “Optimised” are the sample of the version using the results.