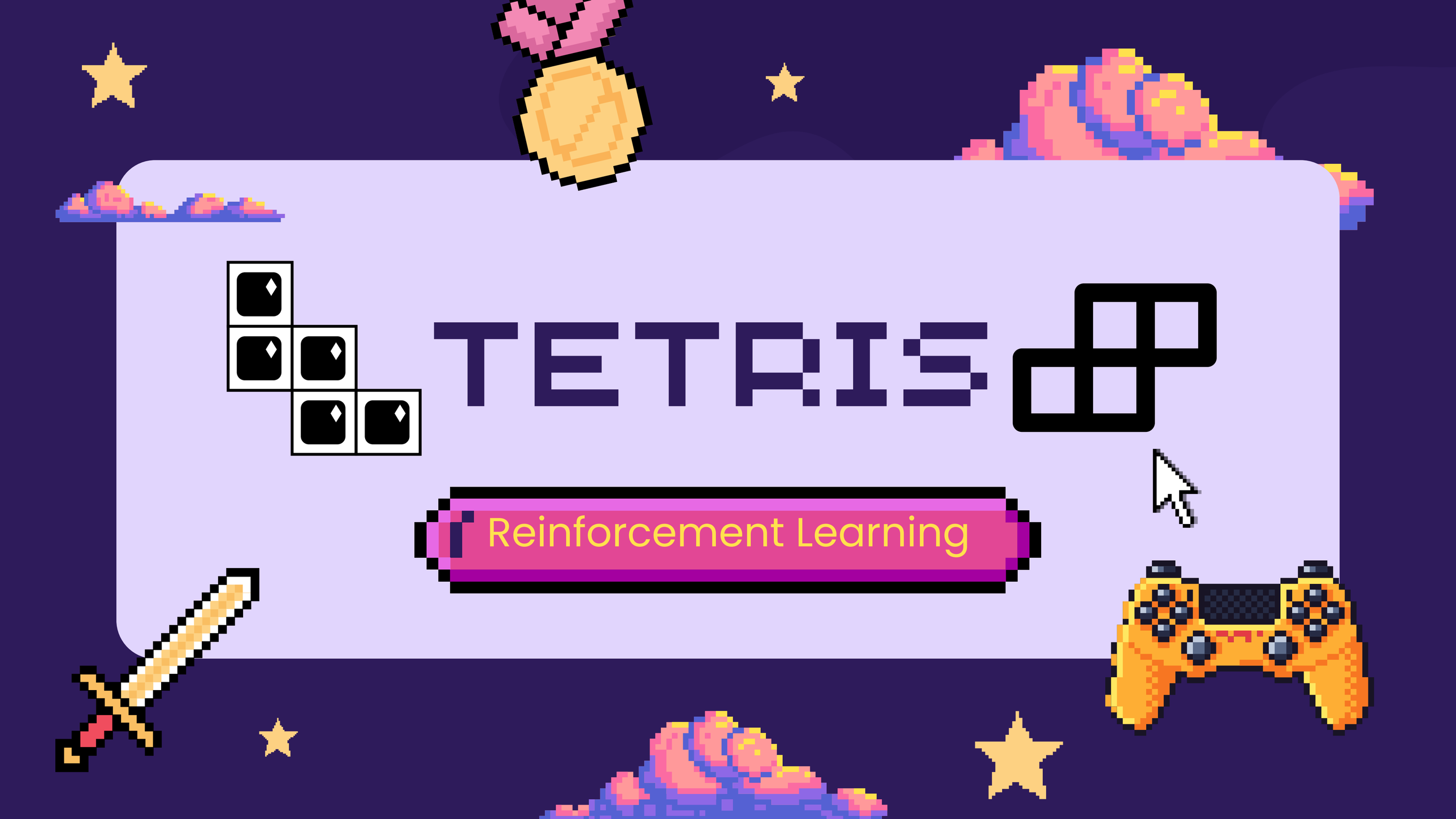
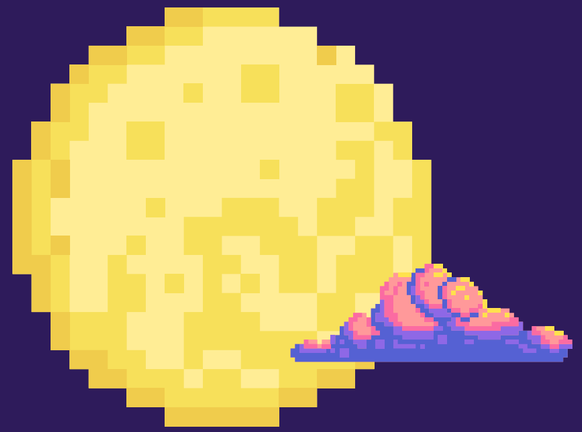
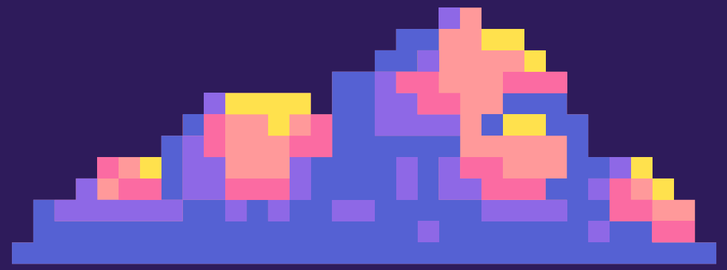

TETRIS
Reinforcement Learning

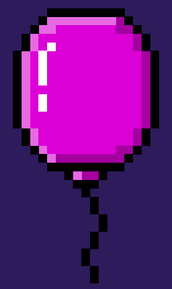
CONTENTS

FOREWORD

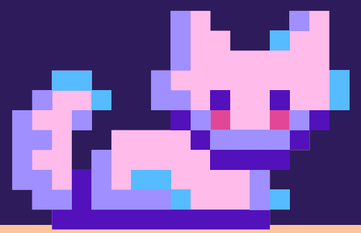
我們對於俄羅斯方塊旋轉方向的隨機性以及消除的方式感到好奇，嘗試使用AI強化學習不斷調整得分的制度和遊戲表現，將訓練後的結果作為資料集，再模擬消除環境的過程中找到許多樂趣。

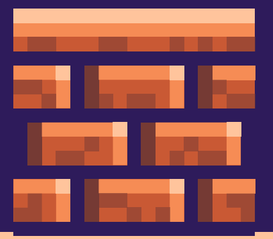


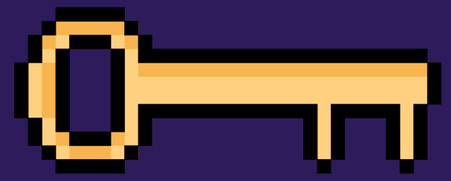




MODEL...







REINFORCEMENT LEARNING

NERUAL NETWORK

DQN

Sarsa

REINFORCEMENT LEARNING

1. 機器學習的一種，電腦透過與動態(dynamic)環境不斷重複互動來學習正確性活動，有助於執行一項任務。
2. 嘗試錯誤(trial-and-error) 學習方法中，電腦在沒有人類干預下運作，主要仰賴動態環境中的資料，例如：天氣或交通流量。

Q-LEARNING...

Q-LEARNING...

AGENT 代理人

Q (STATE, ACTION)

状態 決策

Q LEARNING

Q-table

Action Space

State Space	Q-table	Action (0)	Action (1)	...	Action (m-1)	Action (m)
	0	0	0	0	0	0
	1	0	0	0	0	0
	2	0	0	0	0	0
	...	0	0	0	0	0
	n-1	0	0	0	0	0
	n	0	0	0	0	0

將Q表中的所有值初始化為0，開始訓練

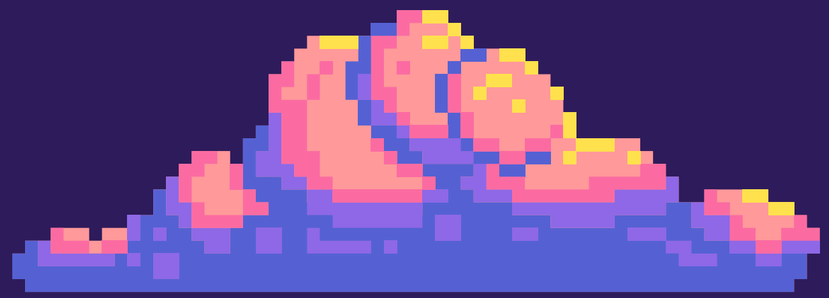

[DEEP Q NETWORK]
NEURAL NETWORK + Q-LEARNING

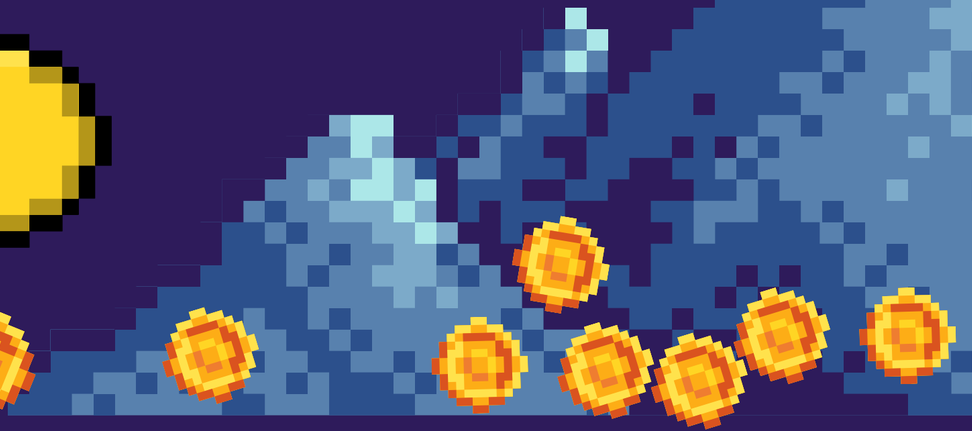
DEEP Q NETWORK (DQN)

Experience replay

Neural Network

Fixed Q-targets






(SARSA)
STATE-ACTION-REWARD-STATE-ACTION

SARSA

Q(St,At) (updated value)

=

Q(St,At) (current value) + a[Rt+1+γ Q(St+1,At+1)-Q(St,At)]

a:learning rate
γ :Discount
St:State St+1:next state
At:Action At+1:next action
Rt+1:Get reward after action

新Q = Q當前 + α (獎勵 + γ *Q預估 - Q當前)

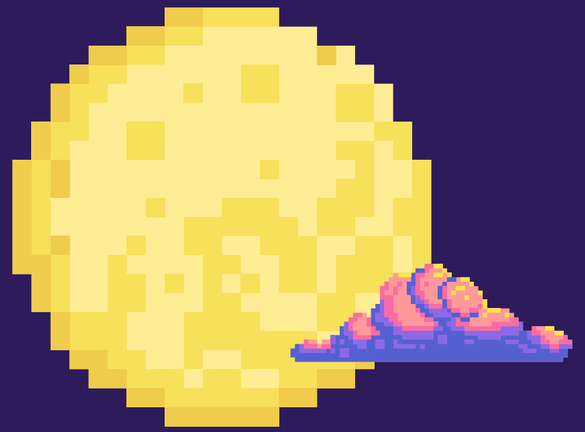

SOURCE...

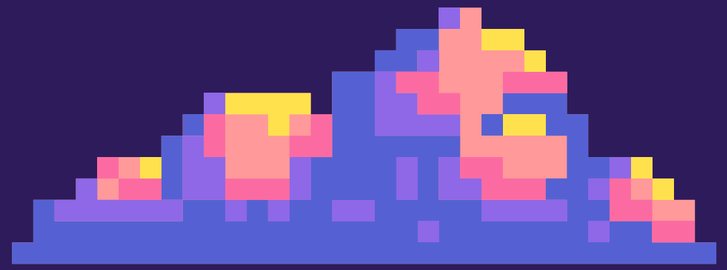
MENU

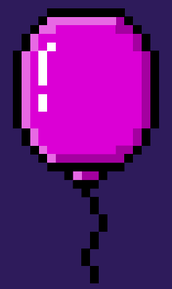
https://youtu.be/RxWS5h1UfI4

https://github.com/michiel-cox/Tetris-DQN

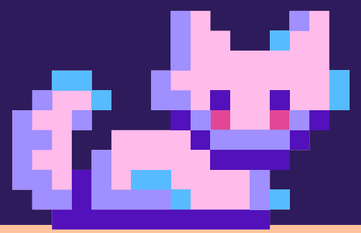
https://github.com/nuno-faria/tetris-ai

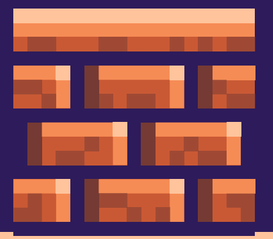




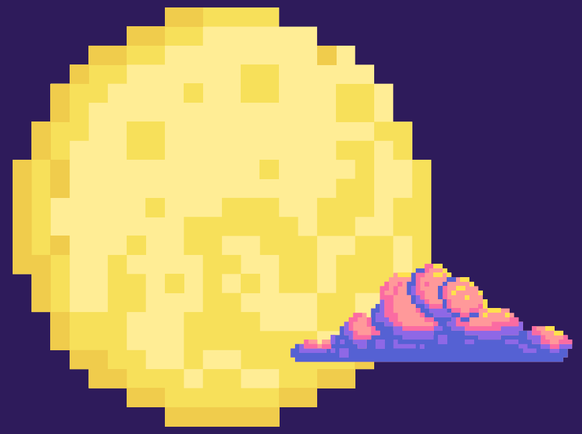


CODE USAGE...



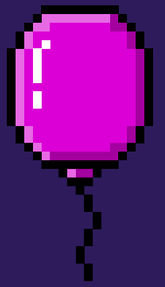


ARCHiTECTURE



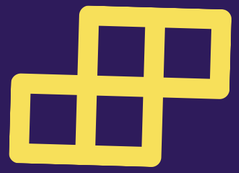
DQN Agent

App

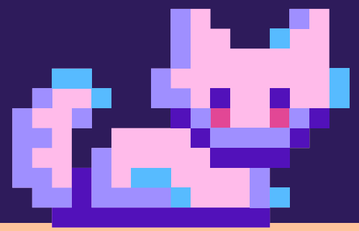


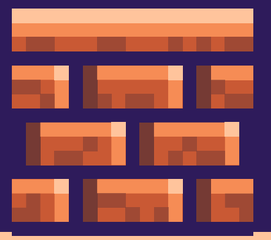
Tetris

Tetromino

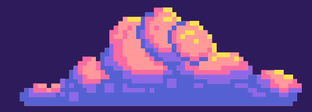


Block

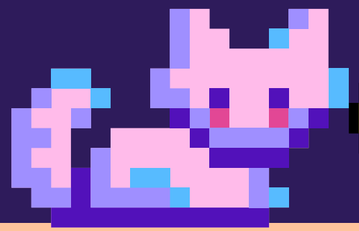




NEERUAL MODEL

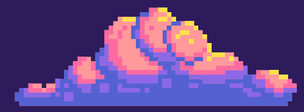


```python
def _build_model(self):
    model = Sequential([
        Dense(64, input_dim=self.state_size, activation='relu'),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1, activation='linear')
    ])

    model.compile(loss='mse', optimizer='adam', metrics=['mean_squared_error'])
    return model
```

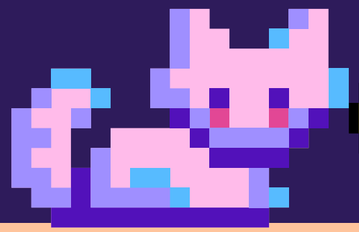


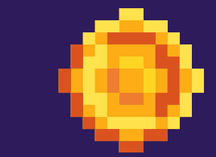
input_dim | 4 | 9

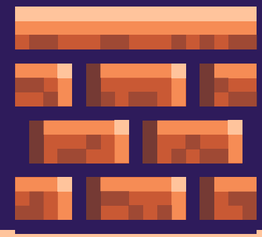
MEMORY REPLAY



```python
class ExperienceBuffer:
    def __init__(self, buffer_size=20000):
        self.buffer = deque(maxlen=buffer_size)

    def add(self, experience):
        self.buffer.append(experience)
```





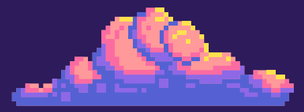


state

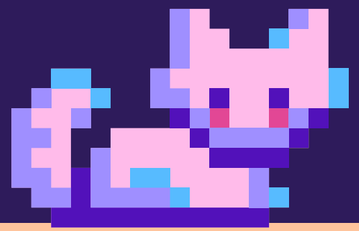
reward

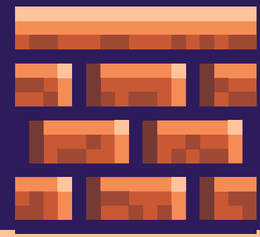
game over

REWARD



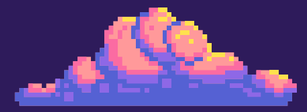
```
score += (10 - self.tetris.landing_height) * 10
self.total_score += score

# Start new round
if self.tetris.game_over:
    if self.episodes >= 1500:
        score -= 200
    else:
        score -= 20
```



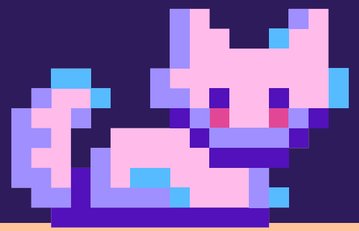


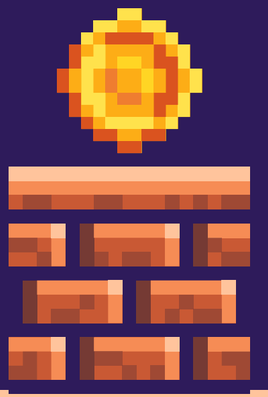
```
self.points_per_line = {0 : 0, 1 : 100, 2 : 300, 3 : 600, 4 : 1200}
```

STATE

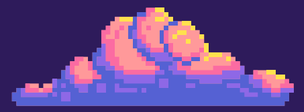


```
""" For all rotations """
for rotation in rotations:
    piece = TETROMINO[piece_id][rotation]
    min_x = min([p[0] for p in piece])
    max_x = max([p[0] for p in piece])

    # For all positions
    for x in range(-min_x, WIDTH - max_x): # -min_x ~ WIDTH-max_x-1 為piece在特定rotation可移動範圍
        pos = [x, 0]

        # Drop piece
        while not self.tetris.is_collide_for_prediction(piece, pos):
            pos[1] += 1
        pos[1] -= 1

        if pos[1] >= 0:
            board = self.tetris.put_tetromino_blocks_in_array_for_prediction(piece, pos)
            new_coords = [(x+pos[0], y+pos[1]) for x, y in piece]
            states[(x, rotation)] = self._get_board_props(board, last_piece_coords=new_coords)
        else:
            states[(0, 0)] = [0 for x in range(9)]
return states
```

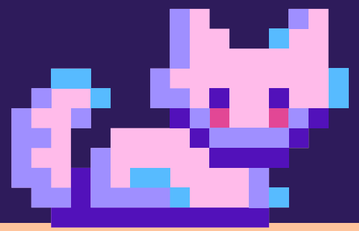




STATE



```
return [
        len(lines),                             # 消除的行數
        total_bumpiness,                        # 每兩直行的高度差總和
        self.tetris.get_hole_count(board),      # 上方有方塊遮擋而產生的空洞
        landing_height,                         # 方塊預計下降的高度
        self.tetris.get_row_transitions(board), # 水平方向的方塊交錯情形
        self.tetris.get_column_transitions(board), # 垂直方向的方塊交錯情形
        self.tetris.get_cumulative_wells(board), # 所有的直行(上方無遮擋方塊)空洞總和
        eroded_piece_cells,                     # 方塊放置後消除的行數 * 方塊本身被消除的數量，EX:消除的1行有2個剛放置的方塊
        self.tetris.get_aggregate_height(board), # 每直行的高度總和
    ]
```



ACTION

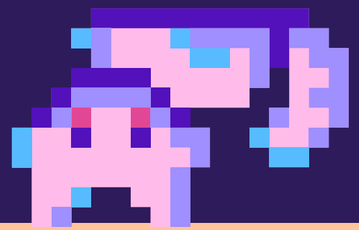
```python
def act(self, state):
    '''Returns the expected score of a certain state'''
    state = np.reshape(state, [1, self.state_size])
    if random.random() <= self.epsilon:
        return self.random_value()
    else:
        return self.predict_value(state)
```

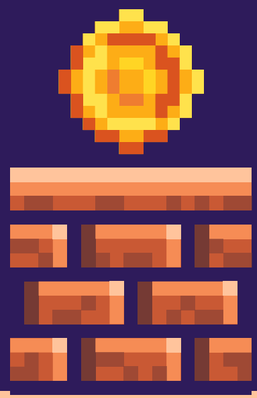
Epsilon : 1

Epsilon_decay : 1/1500

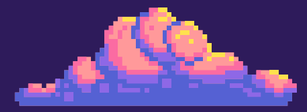
```python
def predict_value(self, state):
    '''Predicts the score for a certain state'''
    return self.model.predict(state)[0]
```

```python
# Update the exploration variable
if self.epsilon > self.epsilon_min:
    self.epsilon -= self.epsilon_decay
```

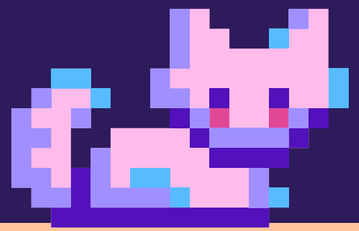


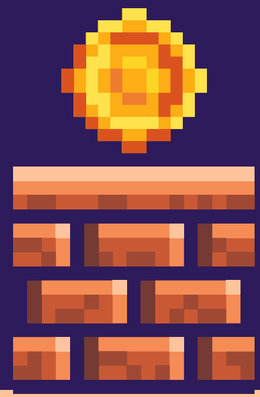


TRAIN

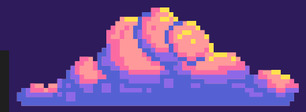


```python
def train(self, episodes=2500):
    scores = []
    result = [['Episode, ', 'Score, ', 'Lines, ', 'Counts\n']]
    for episode in tqdm(range(episodes)):
        current_state = self.env.reset()
        done = False
        steps = 0
        max_steps = 0

        # Game
        self.env.episodes = episode + 1
        while not done and (not max_steps or steps < max_steps):
            self.env.run()
            if self.env.tetris.tetromino.is_already_landing:
                next_states = self.env.get_next_states()
                best_state = self.best_state(next_states.values())

                best_action = None
                for action, state in next_states.items():
                    if state == best_state:
                        best_action = action
                        break
                self.env.new_round()
                reward, done = self.env.play(best_action[0], best_action[1])
                self.add_to_memory(current_state, next_states[best_action], reward, done)
                current_state = next_states[best_action]
                steps += 1

        # Learn
        self.learn(batch_size=512, epochs=1)
```

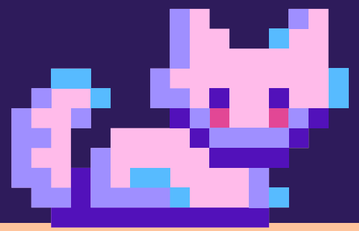


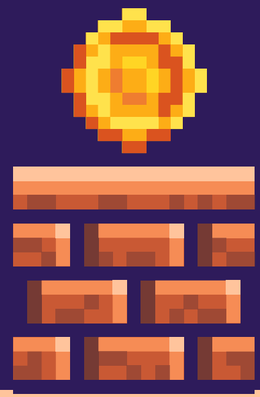


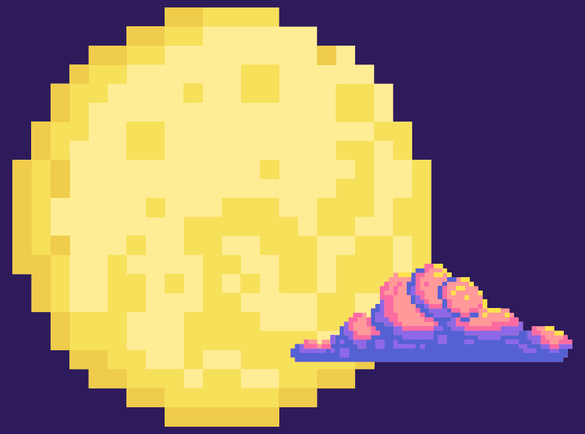
LEARN

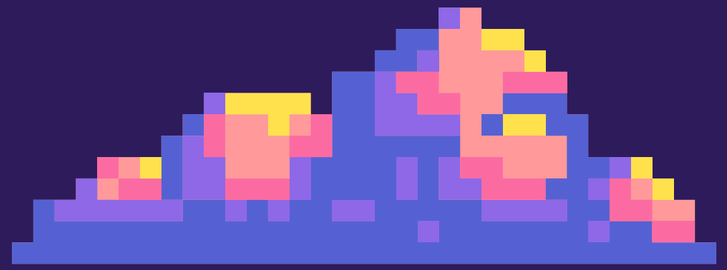


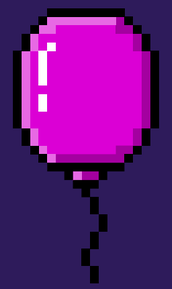
```python
def learn(self, batch_size=32, epochs=3):
    '''Trains the agent'''
    n = len(self.memory.buffer)
    if n >= self.replay_start_size and n >= batch_size:
        batch = random.sample(self.memory.buffer, batch_size) # (current_state, next_state, reward, done)
        # Get the expected score for the next states, in batch (better performance)
        next_states = np.array([x[1] for x in batch])
        next_qs = [x[0] for x in self.model.predict(next_states)]

        x = []
        y = []
        # Build xy structure to fit the model in batch (better performance)
        for i, (state, _, reward, done) in enumerate(batch):
            if not done:
                # Partial Q formula
                new_q = reward + self.discount * next_qs[i]
            else:
                new_q = reward
            x.append(state)
            y.append(new_q)
        # Fit the model to the given values
        self.model.fit(np.array(x), np.array(y), batch_size=batch_size, epochs=epochs, verbose=0)
        # Update the exploration variable
        if self.epsilon > self.epsilon_min:
            self.epsilon -= self.epsilon_decay
```



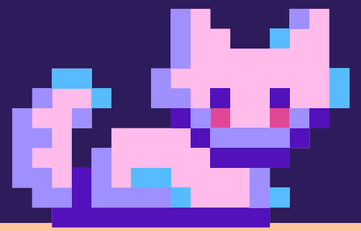


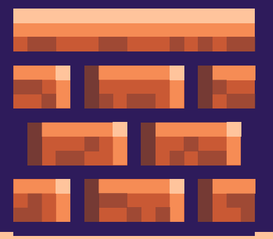


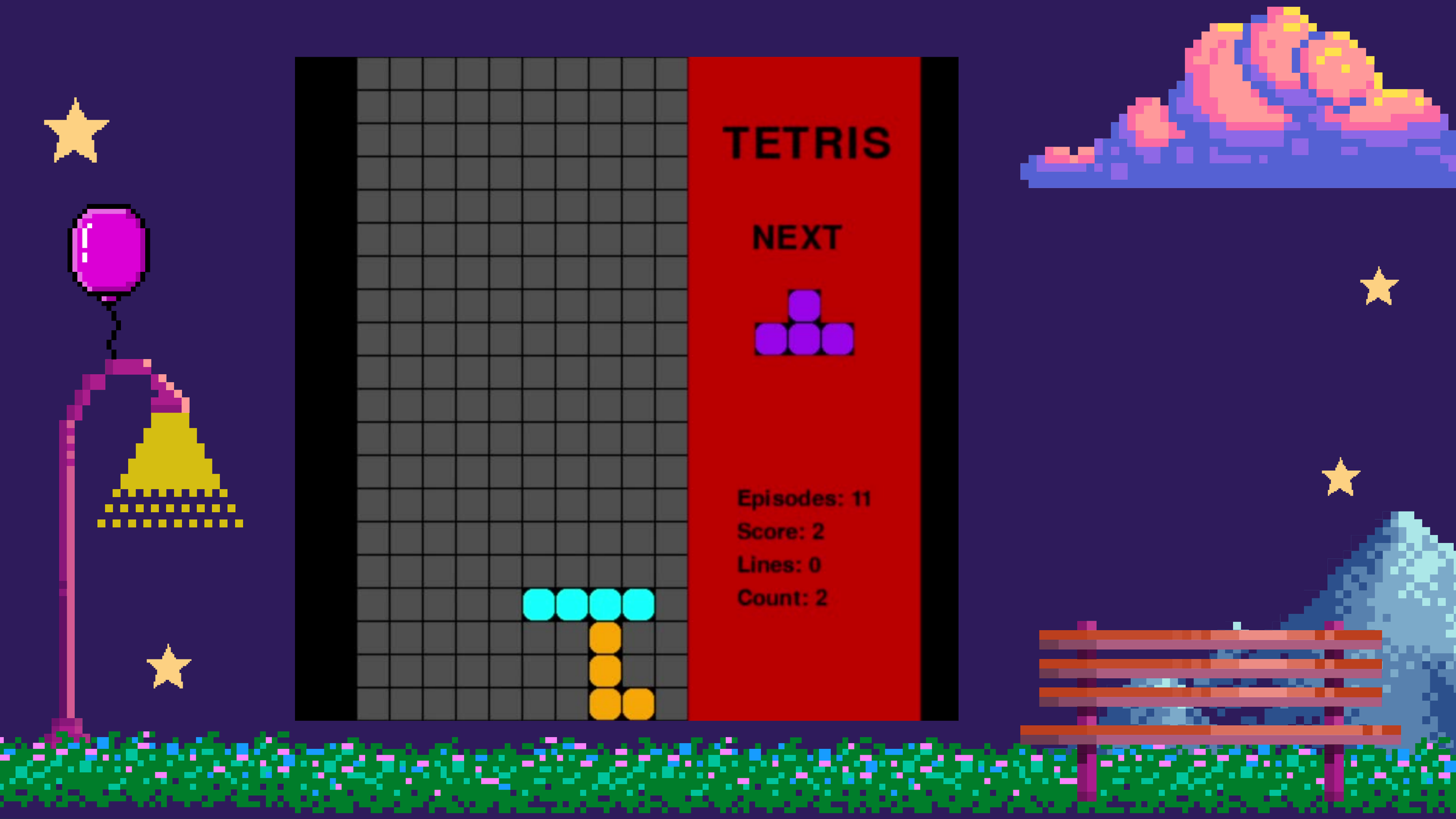


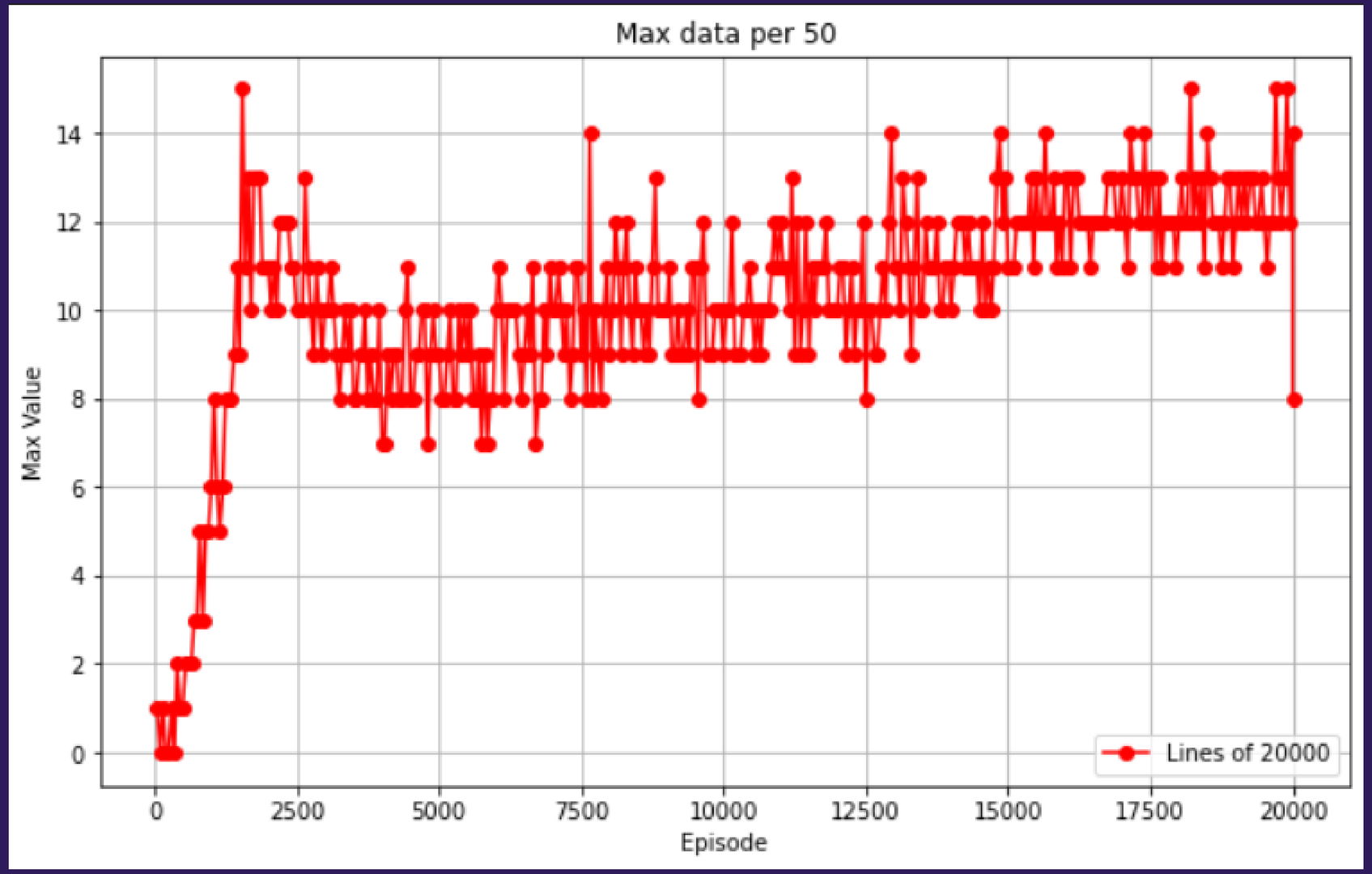


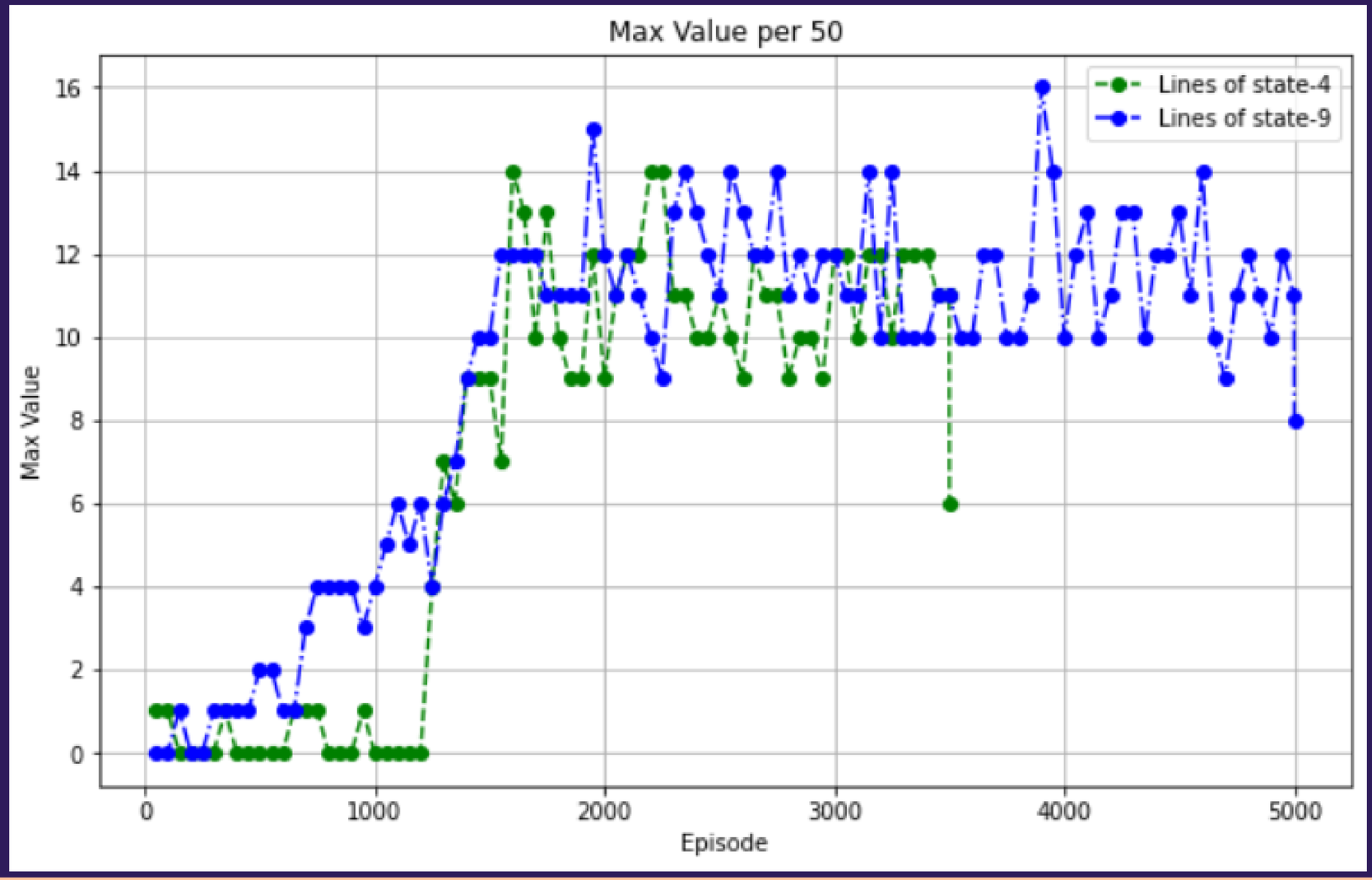
ACHIEVEMENT DISPLAY...

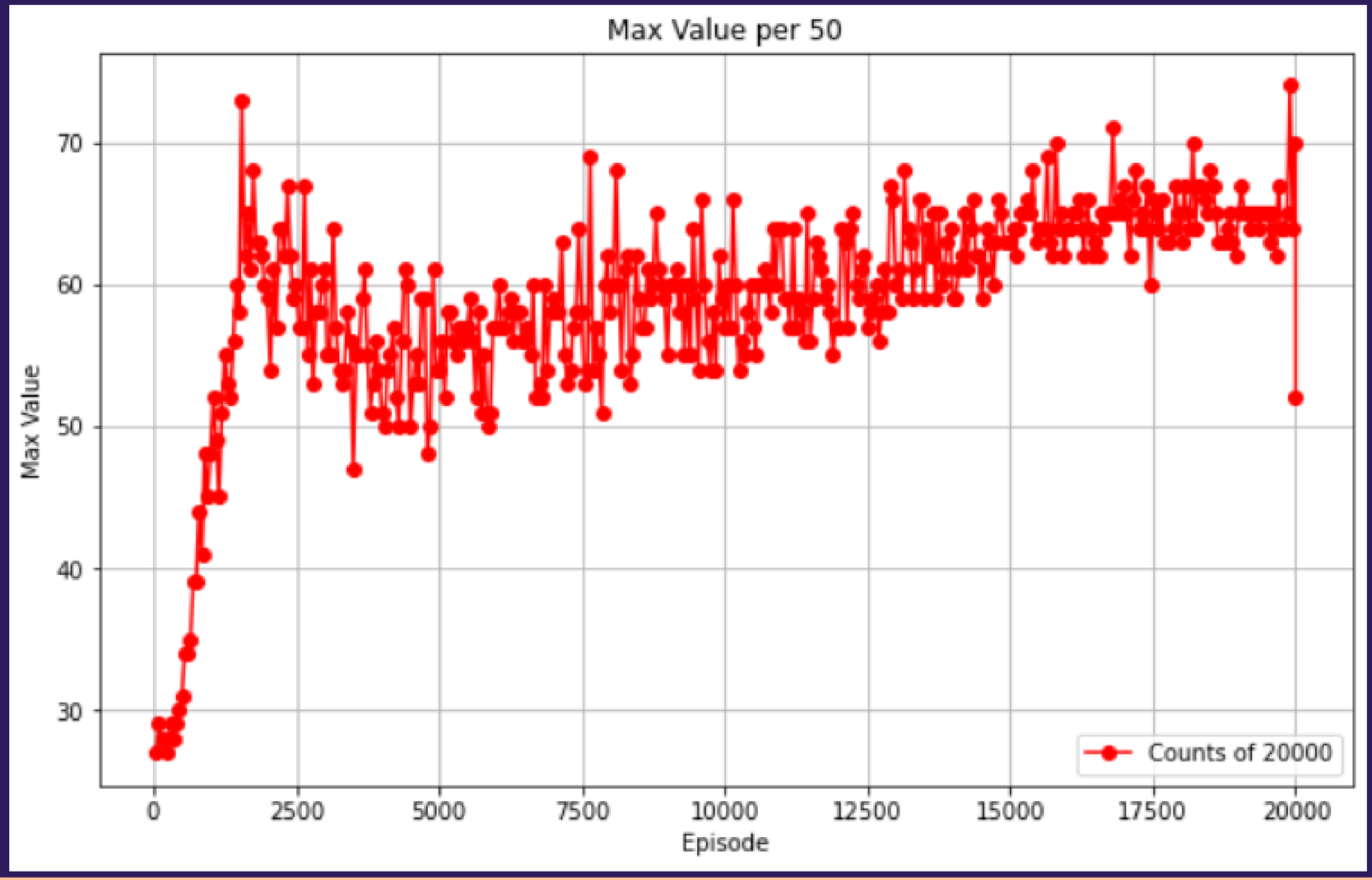


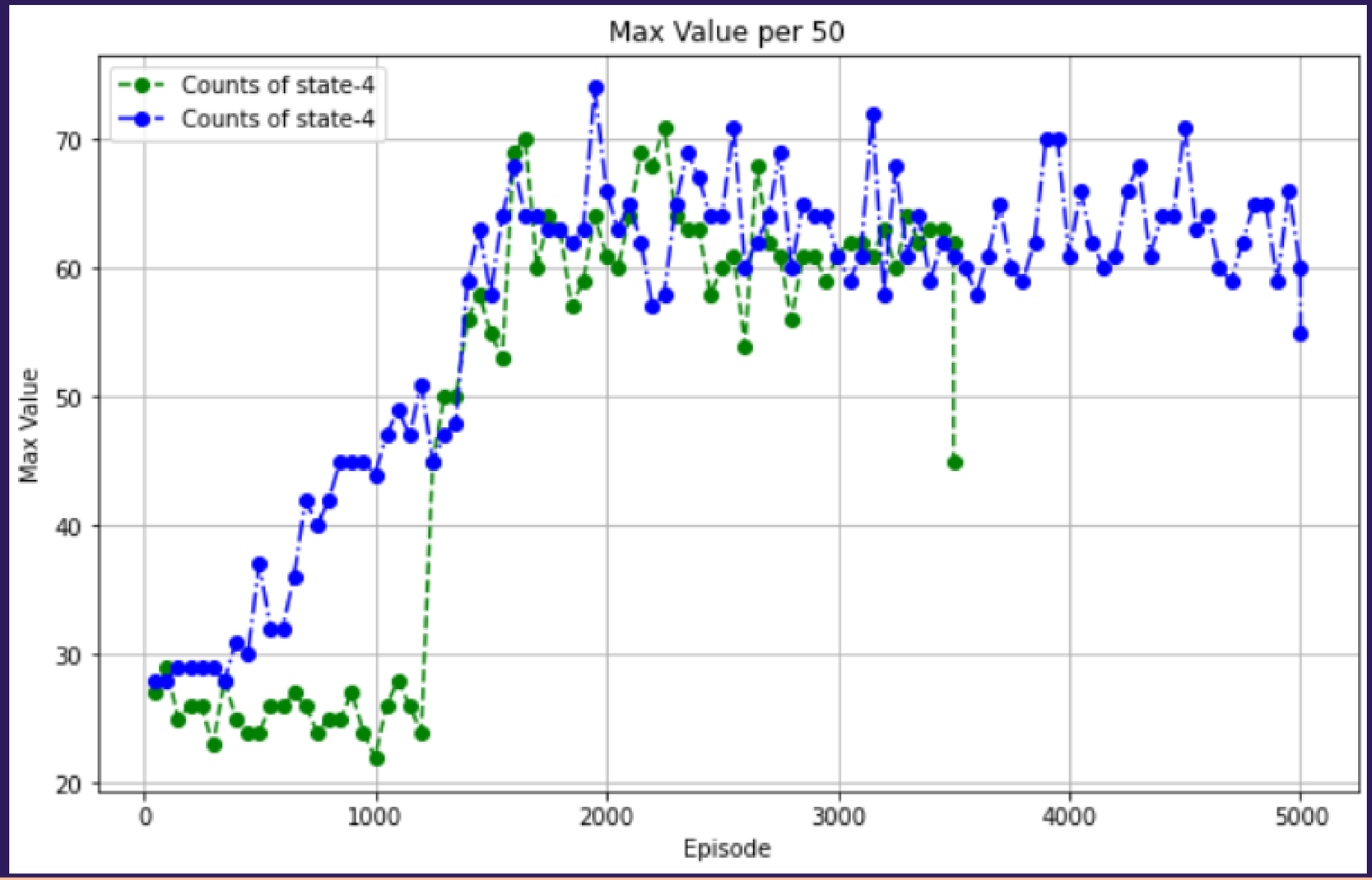



TETRIS
NEXT
Episodes: 11
Score: 2
Lines: 0
Count: 2


Max data per 50
Max Value
Episode
Lines of 20000
0
2
4
6
8
10
12
14
0
2500
5000
7500
10000
12500
15000
17500
20000


Max Value per 50
Lines of state-4
Lines of state-9
Max Value
Episode
0
2
4
6
8
10
12
14
16
0
1000
2000
3000
4000
5000


Max Value per 50
Max Value
70
60
50
40
30
0
2500
5000
7500
10000
12500
15000
17500
20000
Episode
Counts of 20000


Max Value per 50
Counts of state-4
Counts of state-4
Max Value
Episode
20
30
40
50
60
70
0
1000
2000
3000
4000
5000

IMPROVE

1.分數能提高到5000

2.讓AI跟人類1V1對決

3.模型訓練不穩定

例如:訓練第2100次後，訓練分數突然飆高，再來下降，又上升又下降反反覆覆不穩定，直到模型訓練次數達到2450次才平穩，並沒有以線性上升]

IN THE FUTURE

1.讓AI具有長期規劃能力

2.DUELING DQN

3.DOUBLE DQN

4.STATE

EXPERIENCE

- GiT使用
- 大型專案的經驗
- 選擇模型及編譯
- 閱讀他人程式及邏輯



THANK YOU FOR LISTENING