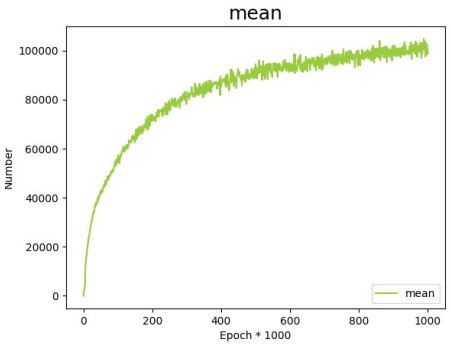
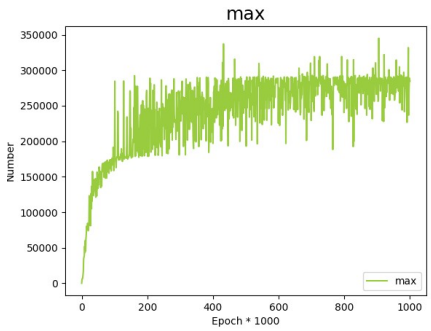
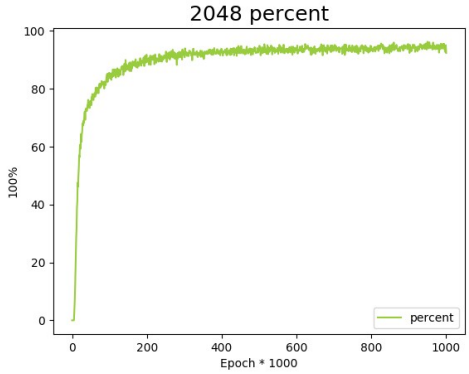
Label 5 :
Temporal Difference Learning

1. 訓練 1,000,000 epoch 時的分數與數據狀況

```
1000000 mean = 101302   max = 286764
         512     100%    (1.9%)
         1024    98.1%   (3%)
         2048    95.1%   (18.6%)
         4096    76.5%   (25.7%)
         8192    50.8%   (50.2%)
         16384   0.6%    (0.6%)
```



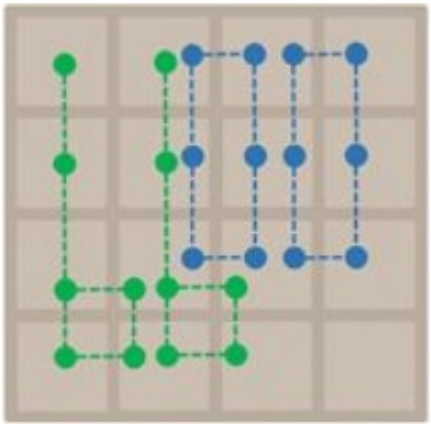




2. 詳細實做介紹:

a. 4 種 6 Tuple:

```
tdl.add_feature(new pattern({0, 4, 8, 12, 13, 9}));
tdl.add_feature(new pattern({1, 5, 9, 13, 14, 10}));
tdl.add_feature(new pattern({1, 5, 9, 10, 6, 2}));
tdl.add_feature(new pattern({2, 6, 10, 11, 7, 3}));
```



b. V(s) 從 after state 的方式改成 before state 後, 變儲存 popup 之後的 state,為紅色框部份, 綠色框是配合 select best move 以及 update episode 時做的 value 運算。

```
b.init();
board beforeBoard = b;
while (true) {
  debug << "state" << std::endl << b;
  state best = tdl.select_best_move(b, n);

  if (best.is_valid()) {
    debug << "best " << best;
    score += best.reward();
    b = best.after_state();
    b.popup();

    best.set_before_state(beforeBoard);
    beforeBoard = b;
    if(!isAfterMode){
      tdl.setStateValue(best);
    }
    path.push_back(best);
  } else {
    best.set_after_state(0);
    best.set_before_state(beforeBoard);
    if(!isAfterMode){
      tdl.setStateValue(best);
    }
    path.push_back(best);
    break;
  }
}
```

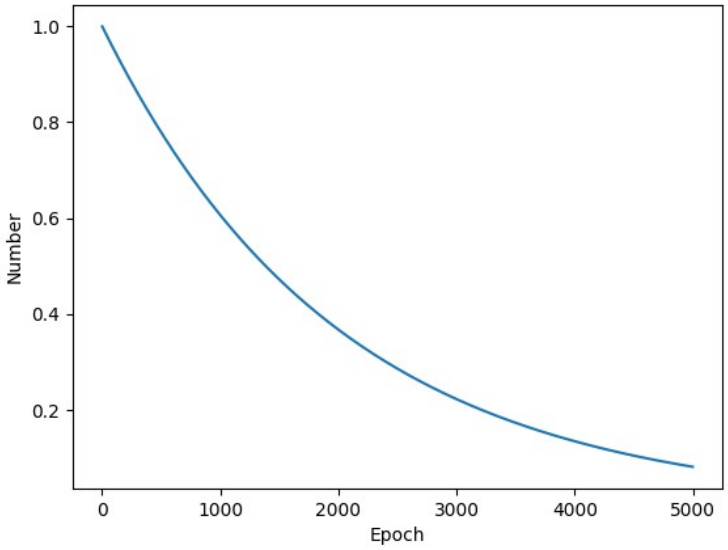
c. select best move 配合 before state 的方式後, 因為有隨機的 popup 的影響, 所以要選這個移動後的盤面好不好就是看他的剩餘空格出現 2 和 4 的期望值, 加上 reward 後選最高的, 為紅框的部份, 也增加看過得盤面的可能, 前 5000epoch 會有一定機率隨機選有效的的步, 為綠框的部份。

```
state select_best_move(const board &b, int _iEpoch) const {
  state after[4] = {0, 1, 2, 3}; // up, right, down, left
  state *best = after;
  vector<int> vTarget;
  int iNumber = -1;
  bool bNeedRandom = _iEpoch <= 5000;
  if (bNeedRandom){
    bNeedRandom = ((double) rand() / RAND_MAX) <= exp(-0.0005  * _iEpoch);
  }
  for (state *move = after; move != after + 4; move++) {
    ++iNumber;
    if (move->assign(b)) {
      if(bNeedRandom){
        vTarget.push_back(iNumber);
      }else{
        if(isAfterMode){
          move->set_value(move->reward() + estimate(move->after_state()));
        }else{
          float boardExpected = getBoardExpected(move->after_state());
          move->set_value(move->reward()+ boardExpected);
        }

        if (move->value() > best->value()){
          best = move;
        }
      }
    } else {
      move->set_value(-std::numeric_limits<float>::max());
    }
    debug << "test " << *move;
  }

  if(bNeedRandom && vTarget.size() > 0){
    iNumber = vTarget[rand() % vTarget.size()];
    state *m = after + iNumber;
    if(isAfterMode){
      m->set_value(m->reward() + estimate(m->after_state()));
    }
    best = m;
  }

  return *best;
}
```



隨機選步的機率下降曲線

d. 計算 state 剩餘空格 2 和 4 期望值的方式, 把 state 剩餘空格各自填入 2 或 4 後, 加總各自 state 的 weight 後, 乘上出現的機率分別是 2 為 0.9, 4 為 0.1, 在除於總格數後即可。

```
float getBoardExpected(const board &b) const{
  board buffer = b;
  float Expected = 0.0f;

  vector<board> rawData = buffer.pre_popup(1);
  float exp = 0.0f;
  for(int i = 0 ; i < rawData.size() ; ++i){
    exp += estimate(rawData[i]);
  }
  exp *= 0.9;
  Expected += exp / rawData.size();

  rawData = buffer.pre_popup(2);
  exp = 0.0f;
  for(int i = 0 ; i < rawData.size() ; ++i){
    exp += estimate(rawData[i]);
  }
  exp *= 0.1;
  Expected += exp / rawData.size();

  Expected /=2;
  return Expected;
}
```

```
vector<board> pre_popup(int _iNumber) {
  vector<board> vectorRaw;

  int space[16], num = 0;
  for (int i = 0; i < 16; i++)
    if (at(i) == 0) {
      space[num++] = i;
    }
  if (num){
    for (int x = 0 ; x < num ; ++x){
      set(space[x], _iNumber);
      vectorRaw.push_back(board(raw));
      set(space[x], 0);
    }
  }
  return vectorRaw;
}
```

e. 更新公式差異在於 before state 是全部都要運算, 不像 after state 會不更新結束的盤面, 以及 reward, 是那移動後的 reward, 而是自己的 reward 做更新。

```
void update_episode(std::vector<state> &path, float alpha = 0.1) const {
  float exact = 0;
  bool pop_back_last = isAfterMode;
  for (/* terminal state */; path.size(); path.pop_back()) {
    if(pop_back_last){
      pop_back_last = false;
      continue;
    }
    state &move = path.back();

    float error = exact - (move.value() - move.reward());

    if(isAfterMode){
      exact = move.reward() + update(move.after_state(), alpha * error);
    }
    else{
      exact = update(move.before_state(), alpha * (move.reward() + error));
    }
  }
}
```

3. 回答問題

a. 在 V(state)中 state 是版面已經有加上隨機 2 或 4 的狀態, V(after state)則是做完一個動作後, 且還沒有加上隨機有 2 或 4 前的狀態。

b. 在 V(state)在選擇動作的時候, 因為出現 2 或 4 是隨機且在空格的位置也是隨機, 所以只能算每個有效動作後, 這個 state 的期望值是多少, 在加上 reward 後取最大的總和, 那 V(after state)就比較單純了, 因為都是不考慮上面的兩種隨機的可能, 可以直接針對 state 的 value 與 reward 相加後取最大的總和。

c. TD-learning,是使用經驗來解決預測問題,且 bootstapping 是每一次改變狀態後, 會取得這次改變狀態的獎勵或回饋, 立即的更新改變狀態前的 value, 也就是說 TD 會更學習的更快更有效率。

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

d. on – policy：統一目標與選擇策略中來學習或改進。
off – policy：目標與選擇的策略不相同,並從中學習或改進。

e. 從這次實驗中發現, before state 在處理期望值時, 會花大量的時間在運算, 同樣要跑到 100,000 是的 after state 只要 50 分鐘左右, before state 則要 220 分鐘, 所以有在想要優化這個期望值得方式, 可能在剩餘格子數多的時候, 可能大於等於 5 格, 是取樣一些格子是 2 或 4 來計算, 反之才全部剩餘格子計算, 可能可以增加計算速度。

4. Test 1000 場 2048 的達成率 95.2%