In [1]:

```
import os
import numpy as np
import pickle
import random
import matplotlib.pyplot as plt
import pandas as pd
```

# 三目並べマシンの学習

勝敗判定器

In [2]:

```
def check_battle(key, my_value):
    """
    盤面の状況が勝負がついたかどうか確認する
    Args:
        key       (str):局面のキー（関数make_keyで作成）
        my_value  (int): { 1：〇、-1：× }
    Returns:
        bool:勝敗 { True：my_valueの勝ち、False：勝負つかず }
    """
    data = np.array(list(map(int, key.split(","))))
    data = data.reshape(3,3)

    # 縦横方向に確認
    for ax in [0,1]:  # 0:横方向に確認、1：縦方向に確認
        data_tmp = np.sum(data == my_value, axis=ax)
        if np.sum(data_tmp == 3) > 0:
            return True  # 勝ち

    # 斜め（対角項）方向に確認
    if np.sum( np.diagonal(data) == my_value) == 3:
        return True  # 勝ち

    # 逆斜め方向に確認
    step = len(data) - 1
    if np.sum( data.ravel()[step:-step:step] == my_value) == 3:
        return True  # 勝ち

    return False
```

局面キー作成関数

In [3]:

```
def make_key(array):
    """
    局面キーを作成する
    Args:
        array  (numpy.array):局面を表すアレイ
    Returns:
        str:局面キー
    """
    array = array.ravel()
    array = array.astype(int)
    return ",".join(map(str, array))
```

局面の重みの更新

In [4]:

```
def edit_weight(key, target_weight, edit_flg):
    """
    局面の重みを更新する
    Args:
        key             (str):局面のキー（関数make_keyで作成）
        target_weight   (int):更新する重みの場所
        edit_flg        (int): {0,1} 0の場合は0に上書き、1の場合は既存値に+1
    Returns:
        (None)
    """
    global learning_weight

    if key not in learning_weight:
        learning_weight[key] = np.ones(9, dtype=int)    # 初期化

    # 重みの更新
    if edit_flg == 0:
        learning_weight[key][target_weight] = 0
    else:
        learning_weight[key][target_weight] += 1
```

局面での指し手の選択

In [5]:

```
def select_position(key, used_position):
    """
    （学習された重みを利用して）局面での指し手を決める
    Args:
        key             (str) :局面のキー（関数make_keyで作成）
        used_position   (list):既に打たれている手の場所
    Returns:
        int:次の指し手 {0～8：次の指し手、-1：次の指し手がない場合}
    """
    global learning_weight

    if key not in learning_weight:
        learning_weight[key] = np.ones(9, dtype=int)    # 初期化

    # 局面の重みを取得
    target_weight = learning_weight[key]
    values = []
    i = 0
    for weight in target_weight:
        if i in used_position:
            weight = 0  # 使用済みの場所を指せないようにする
        values.extend([i]*weight)
        i += 1

    # 指し手の選択
    if len(values) == 0:
        return -1  # 次に選択できる指し手が存在しない場合
    else:
        random.shuffle(values)
        return np.random.choice(values)
```

局面の更新

In [7]:

```
def update_key(key, next_position, updated_value):
    """
    局面（key）に次の指し手を更新する
    Args:
        key             (str):局面キー
        next_position   (int):次の指し手 {0～8}
        updated_value   (int):指した人 {-1:cross, 1:circle}
    Returns:
        str:局面キー
    """
    key_tmp = list(map(int, key.split(",")))
    key_tmp[next_position] = updated_value
    return make_key(np.array(key_tmp))
```

★対戦しながら、学習を行う

In [45]:

```
# 初期化
cross = -1  # 先攻：×
circle = 1  # 後攻：〇
init_value = 0  # 盤面の初期状態
value = [cross, circle]  # circle:〇, cross：×
battle_history = []  # 勝敗履歴
leaning_pattern = []  # 学習済み局面パターン数
# 既存の学習済み重みをクリア
learning_weight = {}
learning_weight.clear()

learning_count = 10000  # 試行回数

# 学習データの読み込み
learning_weight_name = "learning_weight.pickle"
if os.path.exists(learning_weight_name):
    with open(learning_weight_name, mode='rb') as f_ld:
        learning_weight = pickle.load(f_ld)
else:
    # 初期化
    learning_weight = {}
    learning_weight.clear()

print("学習済み局面数（処理前）： {}".format(len(learning_weight)))

for lcount in range(learning_count):  # 試行回数
    # 初期化
    history = []  # 指し手の履歴
    win_flg = False
    key = make_key(np.full(9, init_value))  # 局面
    for i in range(9):  # 最大9回差す
        used_position = list(np.where(np.array(list(map(int, key.split(",")))) != init_value)[0])  #
        if i%2 == 0:
            machine = cross
        else:
            machine = circle

        next_position = select_position(key, used_position)  # 次の指し手の選択
        if next_position == -1:
            # 次の指し手がない場合
            win_flg = True
            battle_history.append(machine*-1)  # 勝敗の履歴
            # 重みの削減（敗者分）
            if len(history) >= 2:
                his_value, his_key ,his_next_pos = history[-2]
                edit_weight(his_key, his_next_pos, 0)  # 学習者の1つ手前の指し手の重みを全て取り去る
            break  # 処理を完了
        else:
            history.append([machine, key, next_position])  # 指し手の履歴を保存
            key = update_key(key, next_position, machine)  # 局面の更新
            if check_battle(key, machine) == True:
                # 勝負が着いた場合
                win_flg = True
                battle_history.append(machine)  # 勝敗の履歴
                # 重みの増加（勝者分）　勝者の全ての指し手の重みを増す
                for his_value, his_key ,his_next_pos in history:
                    if his_value == machine:
```

```
                    edit_weight(his_key, his_next_pos, 1)
                # 重みの削減（敗者分）
                if len(history) >= 2:
                    his_value, his_key ,his_next_pos = history[-2]
                    edit_weight(his_key, his_next_pos, 0)  # 学習者の1つ手前の指し手の重みを全て取り
                break  # 処理を完了
    # 勝敗の履歴
    if win_flg == False:
        battle_history.append(0)
    # 学習済み局面パターン数
    leaning_pattern.append(len(learning_weight))

    if lcount%100 == 0:
        print(".", end="")

print("finish")
print("学習済み局面数（処理後）: {}".format(len(learning_weight)))

# 学習済みデータの保存
with open("learning_weight.pickle", mode="wb") as f_ld:
    pickle.dump(learning_weight, f_ld)

# 勝敗履歴の保存
# 過去の「勝敗履歴」の読み込み
battle_history_name = "battle_history.pickle"
if os.path.exists(battle_history_name):
    with open(battle_history_name, mode='rb') as f_bh:
        battle_history_old = pickle.load(f_bh)
        battle_history_old.extend(battle_history)
        battle_history = battle_history_old
with open("battle_history.pickle", mode='wb') as f_bh:
    pickle.dump(battle_history, f_bh)

# 学習済み局面パターン数の保存
# 過去の「学習済み局面パターン数」の読み込み
leaning_pattern_name = "leaning_pattern_amount.pickle"
if os.path.exists(leaning_pattern_name):
    with open(leaning_pattern_name, mode='rb') as f_lp:
        leaning_pattern_old = pickle.load(f_lp)
        leaning_pattern_old.extend(leaning_pattern)
        leaning_pattern = leaning_pattern_old
with open('leaning_pattern_amount.pickle', mode='wb') as f_lp:
    pickle.dump(leaning_pattern, f_lp)
```

```
学習済み局面数（処理前）: 4343
.....................................................................................
finish
学習済み局面数（処理後）: 4444
```

★勝敗の履歴

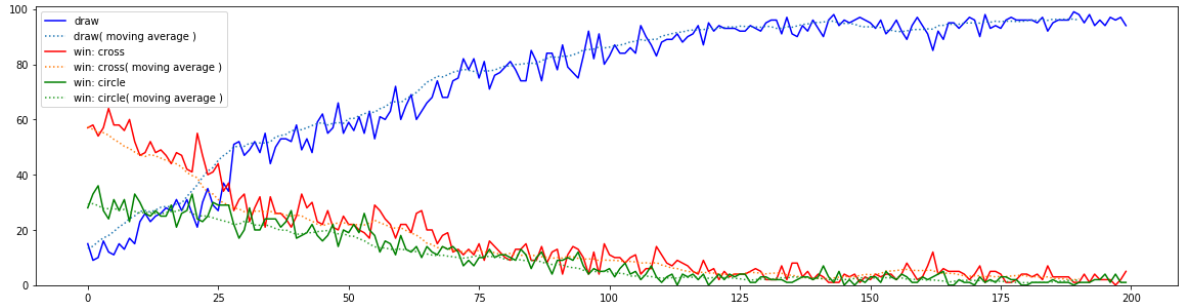
In [46]:

```
v = [0,cross,circle]
v_label = ["draw", "win: cross", "win: circle"]
colors = ["b","r","g"]
y_range = (0,101)

fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(111)

for i in range(3):
    result = np.sum(np.where(np.array(battle_history).reshape((len(battle_history)//100, 100)) == v
    ax.plot(result, label=v_label[i], linestyle="-", color=colors[i] )  # 100回毎の勝率
    ax.plot(np.convolve(result, np.ones(10), mode="valid")/10, label=v_label[i]+"( moving average )"

ax.set_ylim(y_range[0], y_range[1])
ax.legend()
plt.show()
```



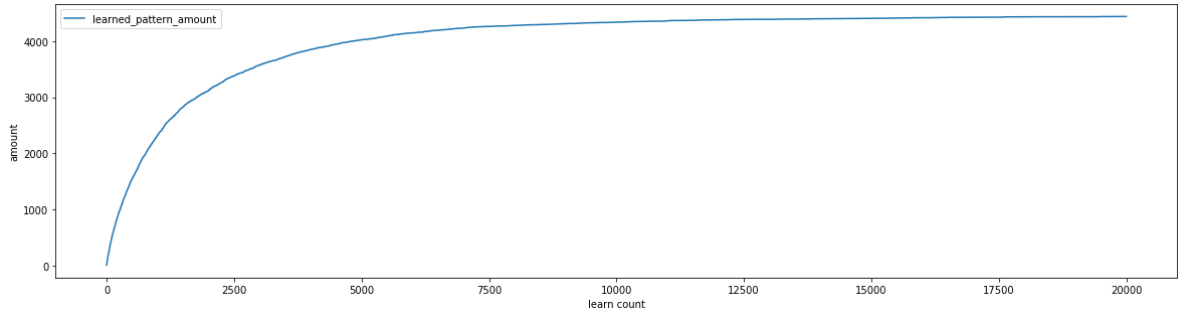

★学習済み局面パターン数の推移

In [48]:

```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(111)

# 学習済み局面パターン数
ax.plot(leaning_pattern, label="learned_pattern_amount", linestyle="-")

ax.set_ylabel("amount")
ax.set_xlabel("learn count")
ax.legend()
plt.show()
```



python2.7の場合、pickleはprotocolが2までが対象なので、改めて出力

In [60]:

```
with open("learning_weight_for_python2_7.pickle", mode="wb") as f_ld:
    pickle.dump(learning_weight, f_ld, protocol=2)
```