In [1]:
```
import numpy as np
import pickle
import random
import matplotlib.pyplot as plt
import pandas as pd
```

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  # 仕様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)
```

In [6]:
```
def select_position_random(used_position):
    """
    （学習された重みを利用せず、ランダムに）局面での指し手を決める
    Args:
        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}
    """
    # 局面の重みを取得
    target_weight = np.ones(9, dtype=int)
    values = []
    i = 0
    for weight in target_weight:
        if i in used_position:
            weight = 0  # 仕様済みの場所を指せないようにする
        values.extend([i]*weight)
        i += 1

    # 差し手の選択
    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

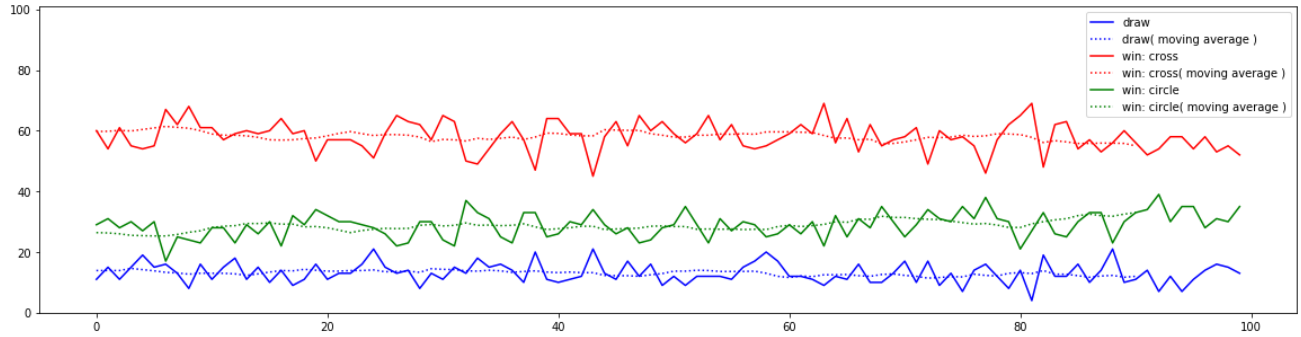
In [7]:
```
def update_key(key, next_position, updated_value):
    """
    局面（key）に次の指し手を更新する
    Args:
        key             (str):局面キー
        next_position   (int):次の指し手 {0～8}
        updated_value   (int):指した人 {-1:cross, 1:circle}
    Returns:
        str:局面キー
    """
    key_tmp = list(map(int, key.split(",")))
    key_tmp[next_position] = updated_value
    return make_key(np.array(key_tmp))
```

未学習状態での勝敗履歴

In [9]:
```
# 初期化
cross = -1  # 先攻：×
circle = 1  # 後手：〇
init_value = 0  # 盤面の初期状態
value = [cross, circle]  # circle:〇, cross：×
battle_history = []  # 勝敗履歴
learning_count = 10000  # 試行回数

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回指す
        used_position = list(np.where(np.array(list(map(int, key.split(",")))) != init_value)[
0])  # 既に打たれている手の場所
        next_position = select_position_random(used_position)  # 次の指し手の選択（未学習、常
にランダム選択）
        if next_position == -1:
            # 次の指し手がない場合
            win_flg = True
            battle_history.append(value[i%2]*-1)  # 勝敗の履歴
            break  # 処理を完了
        else:
            history.append([value[i%2], key, next_position])  # 指し手の履歴を保存
            key = update_key(key, next_position, value[i%2])  # 局面の更新
            if check_battle(key, value[i%2]) == True:
                # 勝負が着いた場合（学習者の勝利時）
                win_flg = True
                battle_history.append(value[i%2])  # 勝敗の履歴
                break  # 処理を完了
    # 勝敗の履歴
    if win_flg == False:
        battle_history.append(0)

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

print("finish")
```

```
..........................................................................................
finish
```

In [10]:
```
v = [0,cross,circle]
v_label = ["draw", "win: cross", "win: circle"]
colors = ["b","r","g","y"]
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[i] ,1, 0), axis=1)
    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 aver
age )", linestyle=":", color=colors[i])  # 100回毎の勝率：移動平均

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



学習

In [17]:
```
"""
--------------------------------------------------------------------------------------------
学習結果を全て消去して、一からやり直したい場合にだけ、このセルを実行してください
※初回は必ず、このセルを実行してください
※2回目以降は、このセルを実行しなかった場合、次のセルで学習を続きから実行できます
--------------------------------------------------------------------------------------------
"""
# 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 = []  # 学習済み局面数
# 既存の学習済み重みをクリア
learning_weight = {}
learning_weight.clear()
```

In [18]:
```
"""
--------------------------------------------------------------------------------
このセルの実行前に、先攻/後攻どちらのマシンを学習させるか選択してください
※先攻を学習させる場合は、後述の変数「target_machine」を「0」に設定してください
※後攻を学習させる場合は、後述の変数「target_machine」を「1」に設定してください
--------------------------------------------------------------------------------
"""
# 初期化
learning_count = 10000  # 試行回数

target_machine = 0  # 学習対象者（0：先攻 or 1：後攻）
#target_machine = 1  # 学習対象者（0：先攻 or 1：後攻）

if target_machine == 0:
    # 学習対象者 = 先攻の場合
    learning_weight_name = "learning_weight_1st.pickle"
    battle_history_name = "battle_history_1st.pickle"
    leaning_pattern_name = "leaning_pattern_name_1st.pickle"
else:
    # 学習対象者 = 後攻の場合
    learning_weight_name = "learning_weight_2nd.pickle"
    battle_history_name = "battle_history_2nd.pickle"
    leaning_pattern_name = "leaning_pattern_name_2nd.pickle"

# 学習データの読み込み
if len(learning_weight) != 0:
    with open(learning_weight_name, mode='rb') as f_ld:
        learning_weight = pickle.load(f_ld)
# 勝敗履歴の読み込み
if len(battle_history) != 0:
    with open(battle_history_name, mode='rb') as f_bh:
        battle_history = pickle.load(f_bh)
# 学習済み局面パターン数の保存
if len(leaning_pattern) != 0:
    with open(leaning_pattern_name, mode='rb') as f_lp:
        leaning_pattern = pickle.load(f_lp)

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])  # 既に打たれている手の場所
        if target_machine == i%2:
            # 学習対象者である場合
            next_position = select_position(key, used_position)  # 次の指し手の選択
            #next_position = select_position_random(used_position)  # 次の指し手の選択（未学
習、常にランダム選択）
            if next_position == -1:
                # 次の指し手がない場合（学習者の敗北）
                win_flg = True
                battle_history.append(value[i%2]*-1)  # 勝敗の履歴（学習者の敗北）
                # 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([value[i%2], key, next_position])  # 指し手の履歴を保存
                key = update_key(key, next_position, value[i%2])  # 局面の更新
                if check_battle(key, value[i%2]) == True:
```

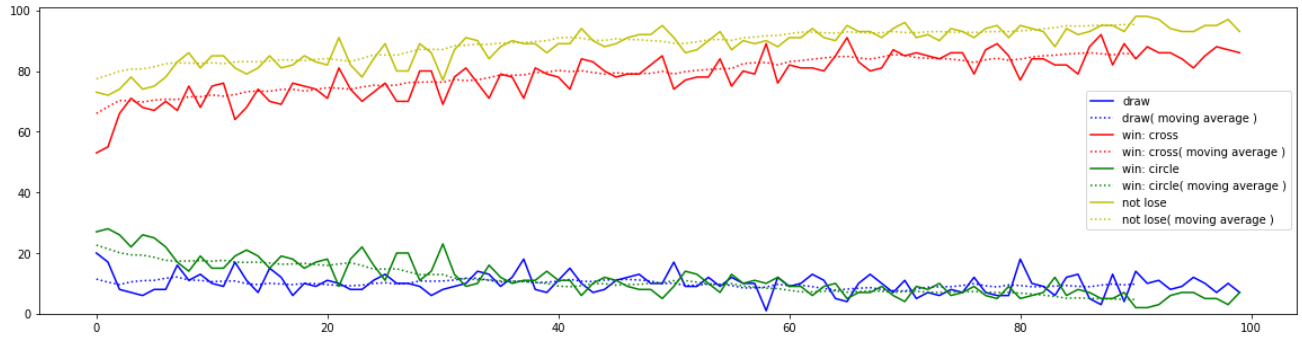
```
                    # 勝負が着いた場合（学習者の勝利時）
                    win_flg = True
                    battle_history.append(value[i%2])  # 勝敗の履歴（学習者の勝利）
                    # 重みの増加（勝者分）　勝者の全ての指し手の重みを増す
                    for his_value, his_key ,his_next_pos in history:
                        if his_value == value[i%2]:
                            edit_weight(his_key, his_next_pos, 1)
                    break  # 処理を完了
        else:
            # 学習対象者でない場合
            next_position = select_position_random(used_position)  # 次の指し手の選択（未学習、常にランダム選択）

            history.append([value[i%2], key, next_position])  # 指し手の履歴を保存
            key = update_key(key, next_position, value[i%2])  # 局面の更新
            if check_battle(key, value[i%2]) == True:
                # 勝負が着いた場合(学習者の敗北時)
                win_flg = True
                battle_history.append(value[i%2])  # 勝敗の履歴（学習者の敗北）
                # 重みの削減（敗者分）
                his_value, his_key ,his_next_pos = history[-2]
                edit_weight(his_key, his_next_pos, 0)  # 敗者（学習者）の最後の指し手の重みを全て取り去る

                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="")

# 学習データの保存
with open(learning_weight_name, mode='wb') as f_ld:
    pickle.dump(learning_weight, f_ld)
# 勝敗履歴の保存
with open(battle_history_name, 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
with open(leaning_pattern_name, mode='wb') as f_lp:
    pickle.dump(leaning_pattern, f_lp)

print("finish")
```

```
.....................................................................................................
finish
```

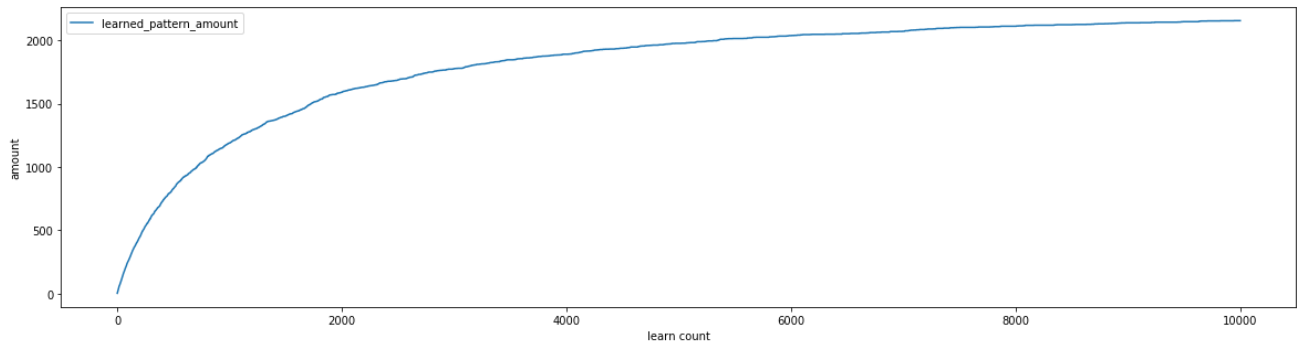
学習状況の確認

先攻のみを学習

In [19]:
```
v = [0,cross,circle]
v_label = ["draw", "win: cross", "win: circle"]
colors = ["b","r","g","y"]
#x_range = (0,150)
#x_range = (0,5000)
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[i] ,1, 0), axis=1)
    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 aver
age )", linestyle=":", color=colors[i])  # 100回毎の勝率：移動平均

# 負けない確率
result = np.sum(np.where(np.array(battle_history).reshape((len(battle_history)//100, 100)) !=
circle ,1, 0), axis=1)
ax.plot(result, label="not lose", linestyle="-", color=colors[3] )  # 100回毎の負けない確率
ax.plot(np.convolve(result, np.ones(10), mode="valid")/10, label="not lose( moving average )",
linestyle=":", color=colors[3])  # 100回毎の負けない確率：移動平均

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

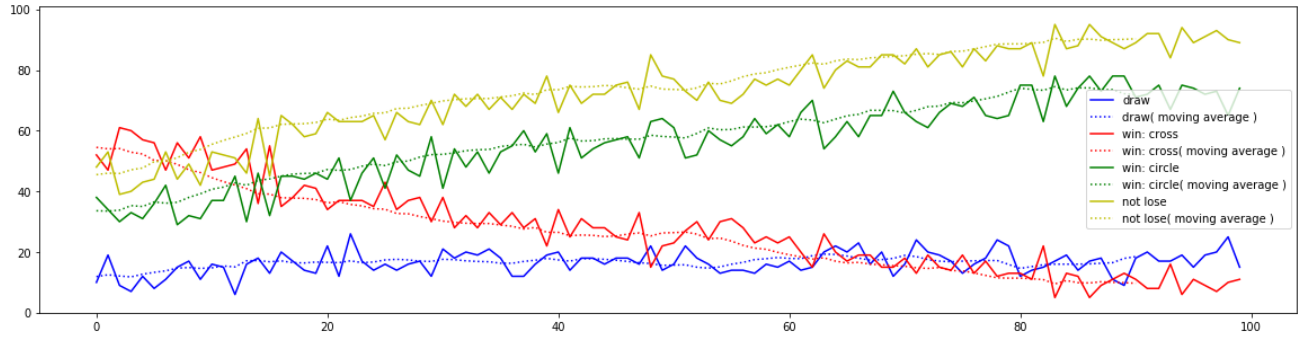


In [20]:
```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(111)

# 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()
```

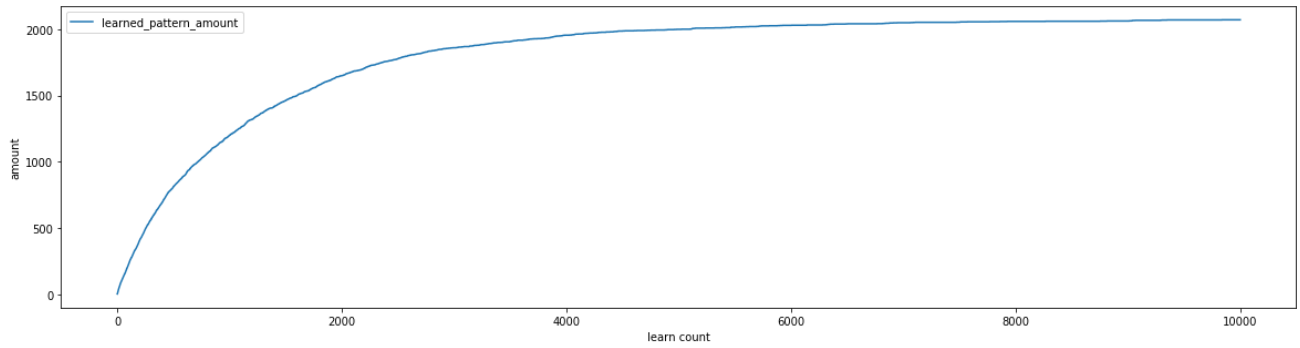


後攻のみを学習

In [13]:
```
v = [0,cross,circle]
v_label = ["draw", "win: cross", "win: circle"]
colors = ["b","r","g","y"]
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[i] ,1, 0), axis=1)
    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 )", linestyle=":", color=colors[i])  # 100回毎の勝率：移動平均

# 負けない確率
result = np.sum(np.where(np.array(battle_history).reshape((len(battle_history)//100, 100)) != cross ,1, 0), axis=1)
ax.plot(result, label="not lose", linestyle="-", color=colors[3] )  # 100回毎の負けない確率
ax.plot(np.convolve(result, np.ones(10), mode="valid")/10, label="not lose( moving average )", linestyle=":", color=colors[3])  # 100回毎の負けない確率：移動平均

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

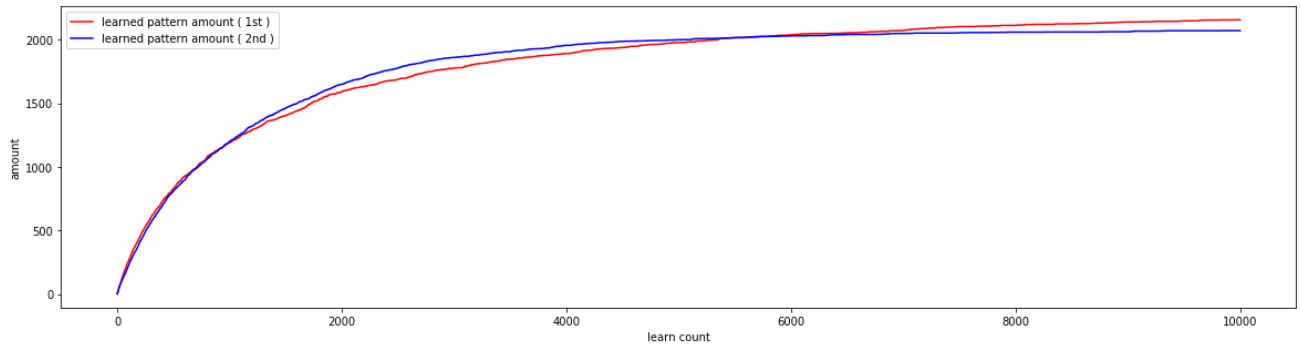


In [14]:
```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(111)

# 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()
```



学習済みの局面パターン数を、先攻/後攻それぞれで確認

In [22]:
```
# 学習済み局面パターン数（先攻）
with open("leaning_pattern_name_1st.pickle", mode='rb') as f_lp:
    leaning_pattern_1st = pickle.load(f_lp)
# 学習済み局面パターン数（後攻）
with open("leaning_pattern_name_2nd.pickle", mode='rb') as f_lp:
    leaning_pattern_2nd = pickle.load(f_lp)

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_1st, label="learned pattern amount ( 1st )", linestyle="-", color="r")
ax.plot(leaning_pattern_2nd, label="learned pattern amount ( 2nd )", linestyle="-", color="b")

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



上手く学習できていたら、keyが重複しないはず

In [24]:
```
# 学習データの読み込み
with open("learning_weight_1st.pickle", mode='rb') as f_ld:
    learning_weight_1st = pickle.load(f_ld)
# 勝敗履歴の読み込み
with open("battle_history_1st.pickle", mode='rb') as f_bh:
    battle_history_1st = pickle.load(f_bh)

# 学習データの読み込み
with open("learning_weight_2nd.pickle", mode='rb') as f_ld:
    learning_weight_2nd = pickle.load(f_ld)
# 勝敗履歴の読み込み
with open("battle_history_2nd.pickle", mode='rb') as f_bh:
    battle_history_2nd = pickle.load(f_bh)

print("学習済みの局面パターン数（先行）: {}".format(len( set(learning_weight_1st.keys()) )))
print("学習済みの局面パターン数（後攻）: {}".format(len( set(learning_weight_2nd.keys()) )))
print("学習済みの局面パターンで、先行と後攻での重複数: {}".format(len(set(learning_weight_1st.
keys()) & set(learning_weight_2nd.keys()))))
print("学習済みの局面パターン数（先行+後攻）: {}".format(len( set(learning_weight_1st.keys())
) + len(set(learning_weight_2nd.keys())) ))
```

```
学習済みの局面パターン数（先行）: 2156
学習済みの局面パターン数（後攻）: 2071
学習済みの局面パターンで、先行と後攻での重複数: 0
学習済みの局面パターン数（先行+後攻）: 4227
```

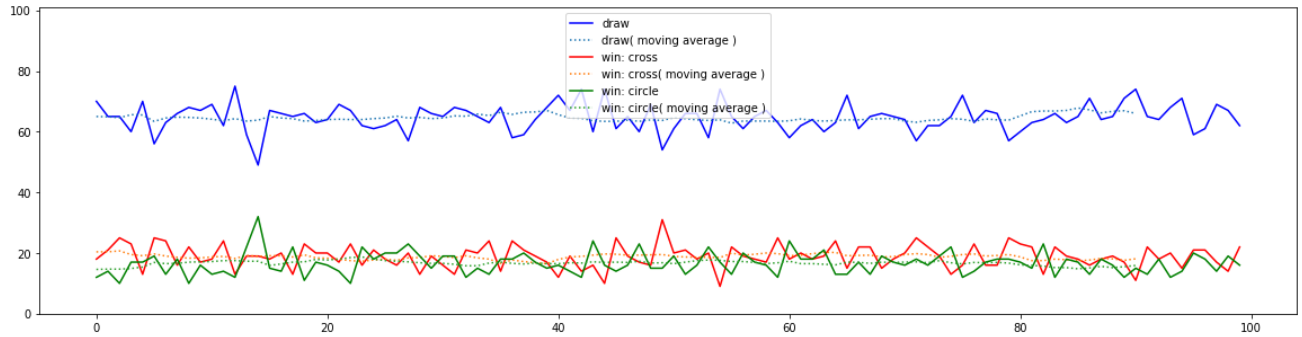
学習済み同士の対決（1）

引き分けばかりになるはず

In [25]:
```
# 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
learning_count = 10000  # 試行回数

# 既存の学習済み重みをクリア
learning_weight = {}
learning_weight.clear()

# 学習データの読み込み
# 学習対象者 = 先攻の場合
with open("learning_weight_1st.pickle", mode='rb') as f_ld:
    learning_weight = pickle.load(f_ld)
# 学習対象者 = 後攻の場合
with open("learning_weight_2nd.pickle", mode='rb') as f_ld:
    learning_weight_add = pickle.load(f_ld)
learning_weight.update(learning_weight_add)

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])  # 既に打たれている手の場所
        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(value[i%2]*-1)  # 勝敗の履歴
            break  # 処理を完了
        else:
            history.append([value[i%2], key, next_position])  # 指し手の履歴を保存
            key = update_key(key, next_position, value[i%2])  # 局面の更新
            if check_battle(key, value[i%2]) == True:
                # 勝負が着いた場合
                win_flg = True
                battle_history.append(value[i%2])  # 勝敗の履歴
                break  # 処理を完了
    # 勝敗の履歴
    if win_flg == False:
        battle_history.append(0)

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

print("finish")
```

```
.....................................................................................
finish
```

In [26]:
```
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[i] ,1, 0), axis=1)
    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 )", linestyle=":")  # 100回毎の勝率：移動平均

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



学習済み同士の対決（2）

対戦しながら、さらに学習を行う

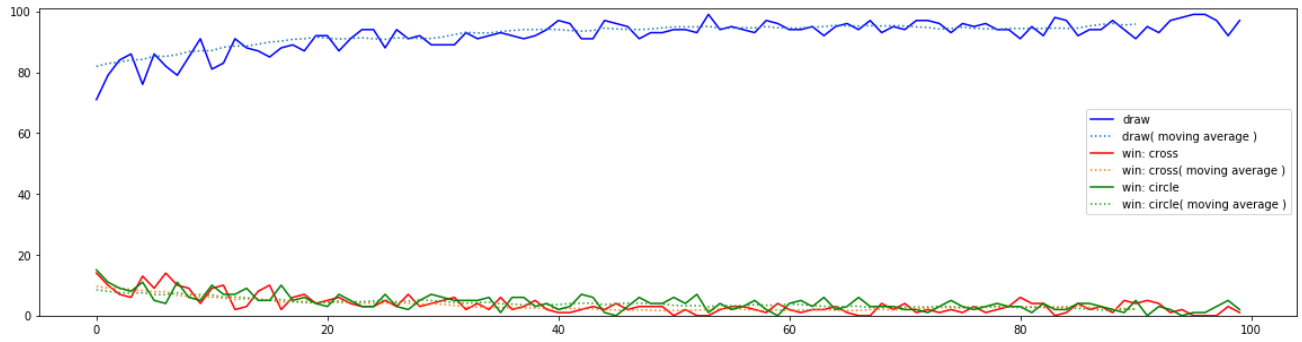
In [31]:
```
# 初期化
cross = -1  # 先攻：×
circle = 1  # 後攻：〇
init_value = 0  # 盤面の初期状態
value = [cross, circle]  # circle:〇, cross：×
battle_history = []  # 勝敗履歴
leaning_pattern = []  # 学習済み局面パターン数
# 既存の学習済み重みをクリア
learning_weight = {}
learning_weight.clear()

learning_count = 10000  # 試行回数

# 学習データの読み込み
# 学習対象者 = 先攻の場合
learning_weight_name = "learning_weight_1st.pickle"
with open(learning_weight_name, mode='rb') as f_ld:
    learning_weight = pickle.load(f_ld)

# 学習対象者 = 後攻の場合
learning_weight_name = "learning_weight_2nd.pickle"
with open(learning_weight_name, mode='rb') as f_ld:
    learning_weight_add = pickle.load(f_ld)
learning_weight.update(learning_weight_add)

# debug
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])  # 既に打たれている手の場所
        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
て取り去る
                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)))
```

```
学習済み局面数（処理前）: 4227
.....................................................................................................
finish
学習済み局面数（処理後）: 4313
```

→ 新たなパターンを少し学習できたようだ

In [32]:
```
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[i] ,1, 0), axis=1)
    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 aver
age )", linestyle=":")  # 100回毎の勝率：移動平均

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



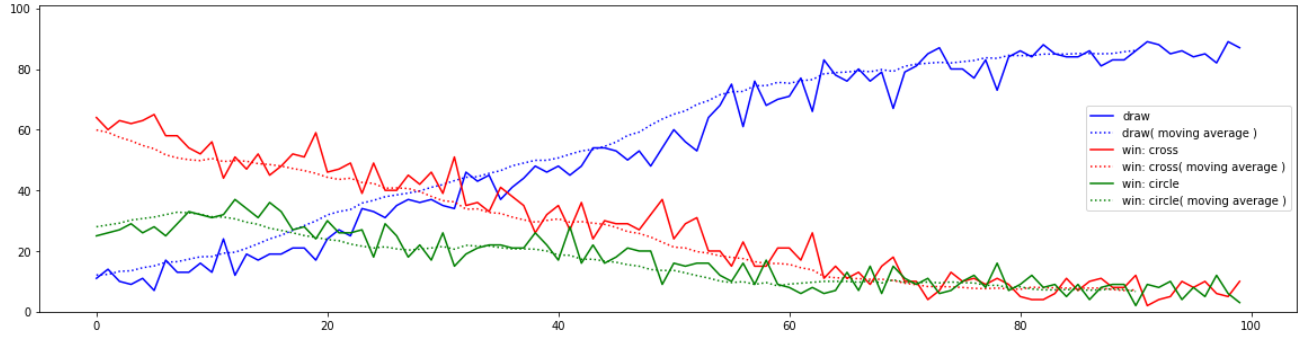
学習済み同士の対決（3）

初めから、対決しながら学習をするようにやり直してみる

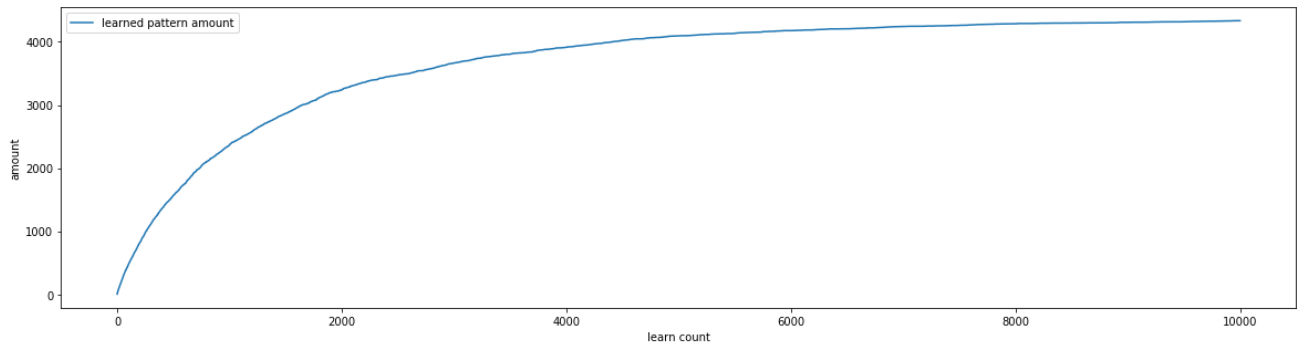
In [33]:
```
# 初期化
cross = -1  # 先攻：×
circle = 1  # 後攻：〇
init_value = 0  # 盤面の初期状態
value = [cross, circle]  # circle:〇, cross：×
battle_history = []  # 勝敗履歴
learning_count = 10000  # 試行回数
# 既存の学習済み重みをクリア
learning_weight = {}
learning_weight.clear()
leaning_pattern = []  # 学習済み局面パターン数

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])  # 既に打たれている手の場所
        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:
            # 次の差し手がない場合
            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
て取り去る
                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")
```

```
finish
```

In [34]:
```
v = [0,cross,circle]
v_label = ["draw", "win: cross", "win: circle"]
colors = ["b","r","g","k"]
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[i] ,1, 0), axis=1)
    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 aver
age )", linestyle=":", color=colors[i])  # 100回毎の勝率：移動平均

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

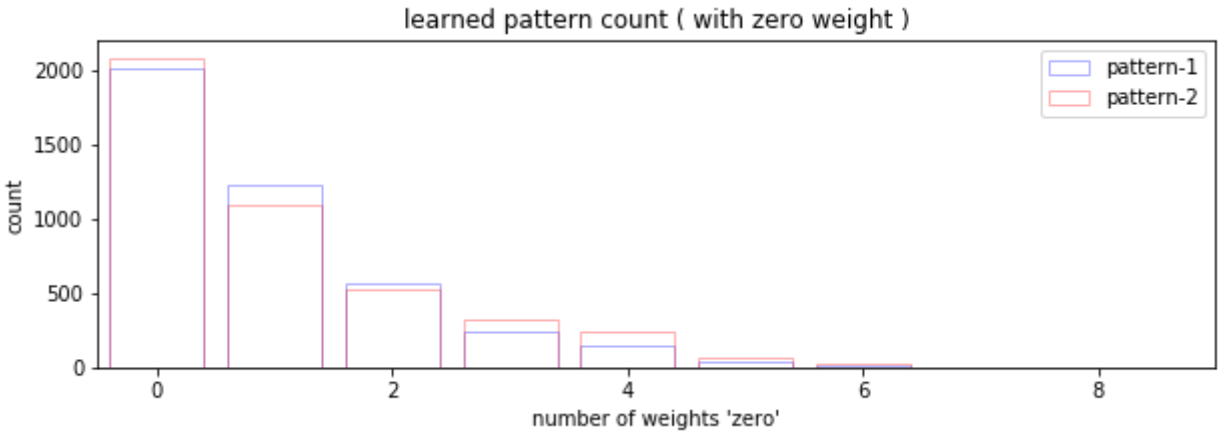


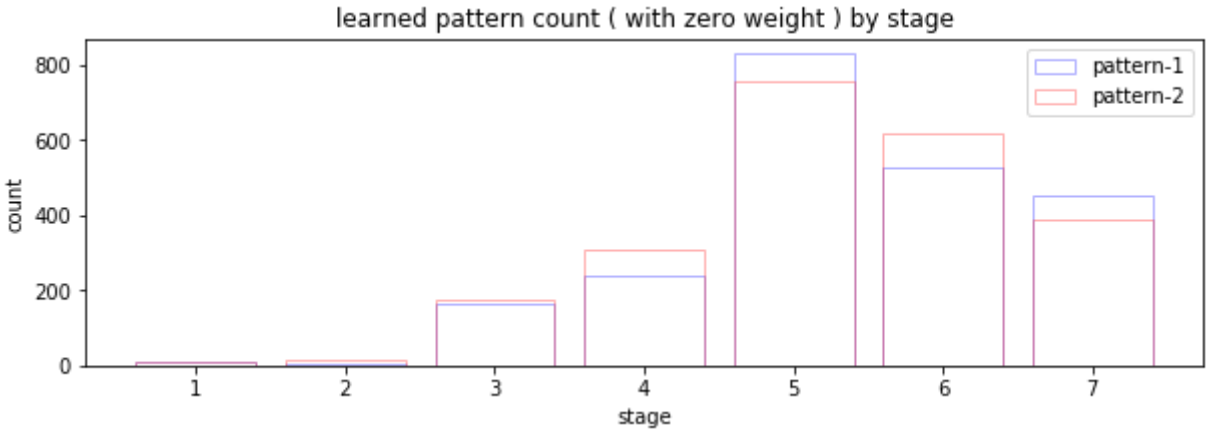
In [36]:
```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(111)

# 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()
```



先攻と後攻を個別学習したものと、同時に学習した結果を比較してみる

In [41]:
```
# ------------------------------
# 先攻と後攻を個別学習した重み
# ------------------------------
# 学習データの読み込み
# 学習対象者 = 先攻の場合
with open("learning_weight_1st.pickle", mode='rb') as f_ld:
    weight_tmp = pickle.load(f_ld)
# 学習対象者 = 後攻の場合
with open("learning_weight_2nd.pickle", mode='rb') as f_ld:
    weight_tmp_add = pickle.load(f_ld)
weight_tmp.update(weight_tmp_add)

df_tmp_1 = pd.io.json.json_normalize(weight_tmp).T
df_tmp_1.columns = ["weight"]
df_tmp_1["zero_count"] = df_tmp_1["weight"].apply(lambda x: np.sum(x == 0))
df_tmp_1["stage"] = [ 9 - list(map(int, x.split(","))).count(0) for x in np.array(df_tmp_1.ind
ex)]

# ------------------------------
# 先攻と後攻を同時に学習した重み
# ------------------------------
df_tmp_2 = pd.io.json.json_normalize(learning_weight).T
df_tmp_2.columns = ["weight"]
df_tmp_2["zero_count"] = df_tmp_2["weight"].apply(lambda x: np.sum(x == 0))
df_tmp_2["stage"] = [ 9 - list(map(int, x.split(","))).count(0) for x in np.array(df_tmp_2.ind
ex)]

# 描画 -------------------------

# 各局面において、重みが「0」となった局面の数
fig = plt.figure(figsize=(10,3))
ax = fig.add_subplot(111)
ax.set_xlim(-0.5, 9)
ax.set_ylim(0, 2200)

tmp = df_tmp_1.groupby("zero_count")["weight"].count()
ax.bar(tmp.index, tmp.values, alpha=0.3, facecolor="None", edgecolor="b", label="pattern-1")
tmp = df_tmp_2.groupby("zero_count")["weight"].count()
ax.bar(tmp.index, tmp.values, alpha=0.3, facecolor="None", edgecolor="r", label="pattern-2")

ax.legend()
ax.set_ylabel("count")
ax.set_xlabel("number of weights 'zero'")
ax.set_title("learned pattern count ( with zero weight )")
plt.show()

# 初手から7手目までの、各手数毎の「重み「0」を持つ局面の数」
fig = plt.figure(figsize=(10,3))
ax = fig.add_subplot(111)
tmp = df_tmp_1[ df_tmp_1.zero_count != 0 ].groupby("stage")["zero_count"].count()
ax.bar(tmp.index, tmp.values, alpha=0.3, facecolor="None", edgecolor="b", label="pattern-1")
tmp = df_tmp_2[ df_tmp_2.zero_count != 0 ].groupby("stage")["zero_count"].count()
ax.bar(tmp.index, tmp.values, alpha=0.3, facecolor="None", edgecolor="r", label="pattern-2")
ax.legend()
ax.set_ylabel("count")
ax.set_xlabel("stage")
ax.set_title("learned pattern count ( with zero weight ) by stage")
plt.show()
```





1手目の推奨となる指し手

(1) 先攻と後攻を個別学習した場合

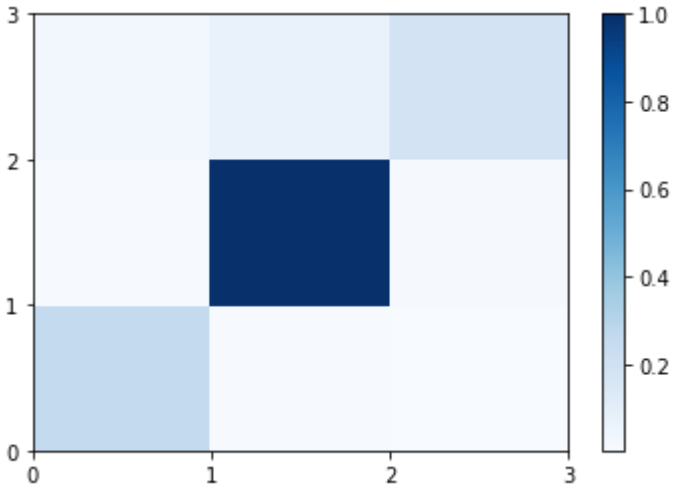
In [42]:
```
df_tmp_1[ df_tmp_1.stage == 0 ]
```

Out[42]:

	weight	zero_count	stage
0,0,0,0,0,0,0,0,0	[1282, 48, 16, 46, 4948, 82, 154, 369, 922]	0	0

In [43]:
```
tmp = np.array(df_tmp_1[ df_tmp_1.index == make_key(np.zeros(9))]["weight"][0]).reshape(3,3)
tmp = tmp / tmp.max()
image = plt.pcolor(tmp, cmap=plt.cm.Blues)
plt.xticks([0,1,2,3])
plt.yticks([0,1,2,3])
plt.colorbar(image)
```

Out[43]: <matplotlib.colorbar.Colorbar at 0x1b1f5c01d30>



(2) 先攻と後攻を同時に学習した場合

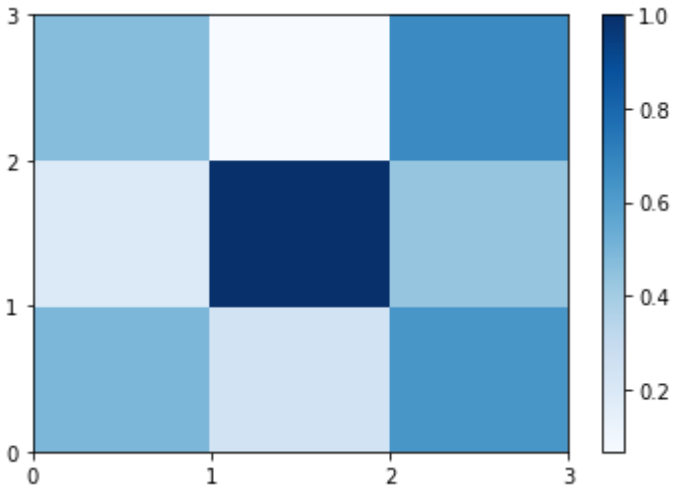
In [44]:
```
df_tmp_2[ df_tmp_2.stage == 0 ]
```

Out[44]:

	weight	zero_count	stage
0,0,0,0,0,0,0,0,0	[284, 140, 362, 112, 573, 251, 268, 38, 387]	0	0

In [45]:
```
tmp = np.array(df_tmp_2[ df_tmp_2.index == make_key(np.zeros(9))]["weight"][0]).reshape(3,3)
tmp = tmp / tmp.max()
image = plt.pcolor(tmp, cmap=plt.cm.Blues)
plt.xticks([0,1,2,3])
plt.yticks([0,1,2,3])
plt.colorbar(image)
```

Out[45]: <matplotlib.colorbar.Colorbar at 0x1b1f5d70ac8>



2手目の推奨となる指し手

(1) 先攻と後攻を個別学習した場合

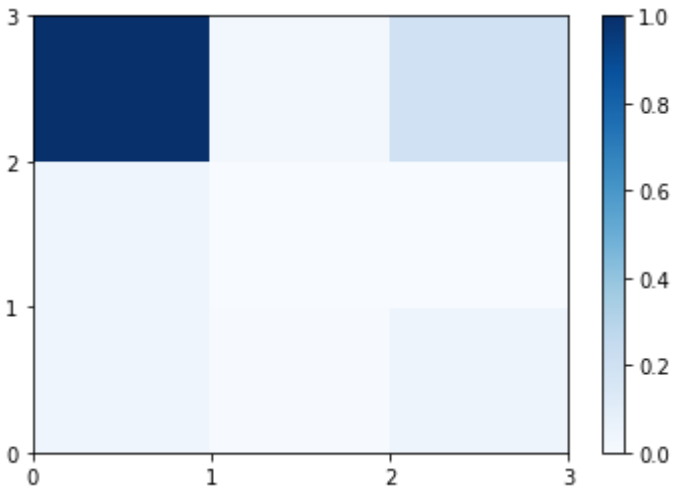
In [46]:
```
df_tmp_1[ df_tmp_1.stage == 1 ]
```

Out[46]:

	weight	zero_count	stage
-1,0,0,0,0,0,0,0,0	[1, 19, 0, 0, 350, 38, 0, 17, 13]	3	1
0,-1,0,0,0,0,0,0,0	[80, 1, 212, 34, 104, 0, 0, 16, 41]	2	1
0,0,-1,0,0,0,0,0,0	[0, 0, 1, 0, 181, 0, 29, 58, 0]	5	1
0,0,0,-1,0,0,0,0,0	[3, 8, 0, 1, 317, 9, 30, 23, 129]	1	1
0,0,0,0,-1,0,0,0,0	[19, 3, 20, 17, 1, 0, 387, 12, 80]	1	1
0,0,0,0,0,-1,0,0,0	[0, 17, 40, 1, 298, 1, 146, 1, 180]	1	1
0,0,0,0,0,0,-1,0,0	[0, 0, 0, 22, 265, 0, 1, 13, 0]	5	1
0,0,0,0,0,0,0,-1,0	[0, 4, 387, 5, 25, 103, 10, 1, 48]	1	1
0,0,0,0,0,0,0,0,-1	[0, 0, 19, 0, 412, 4, 64, 6, 1]	3	1

In [47]:
```
tmp = np.array(df_tmp_1[ df_tmp_1.index == "0,0,0,0,-1,0,0,0,0"]["weight"][0]).reshape(3,3)
tmp = tmp / tmp.max()
image = plt.pcolor(tmp, cmap=plt.cm.Blues)
plt.xticks([0,1,2,3])
plt.yticks([0,1,2,3])
plt.colorbar(image)
```

Out[47]: <matplotlib.colorbar.Colorbar at 0x1b1f6155588>



(2) 先攻と後攻を同時に学習した場合

In [48]:
```
df_tmp_2[ df_tmp_2.stage == 1 ]
```

Out[48]:

	weight	zero_count	stage
-1,0,0,0,0,0,0,0,0	[1, 0, 0, 3, 53, 0, 0, 0, 0]	6	1
0,-1,0,0,0,0,0,0,0	[37, 1, 5, 0, 38, 0, 0, 6, 8]	3	1
0,0,-1,0,0,0,0,0,0	[0, 0, 1, 0, 41, 0, 0, 0, 0]	7	1
0,0,0,-1,0,0,0,0,0	[7, 21, 0, 1, 18, 4, 15, 6, 27]	1	1
0,0,0,0,-1,0,0,0,0	[42, 0, 73, 0, 1, 0, 39, 0, 35]	4	1
0,0,0,0,0,-1,0,0,0	[0, 0, 45, 9, 37, 1, 0, 0, 25]	4	1
0,0,0,0,0,0,-1,0,0	[0, 0, 0, 0, 96, 0, 1, 0, 0]	7	1
0,0,0,0,0,0,0,-1,0	[4, 1, 17, 2, 1, 2, 9, 1, 4]	0	1
0,0,0,0,0,0,0,0,-1	[0, 0, 0, 0, 94, 0, 0, 0, 1]	7	1

In [49]:
```
tmp = np.array(df_tmp_2[ df_tmp_2.index == "0,0,0,0,-1,0,0,0,0"]["weight"][0]).reshape(3,3)
tmp = tmp / tmp.max()
image = plt.pcolor(tmp, cmap=plt.cm.Blues)
plt.xticks([0,1,2,3])
plt.yticks([0,1,2,3])
plt.colorbar(image)
```

Out[49]: <matplotlib.colorbar.Colorbar at 0x1b1f5e5cc88>