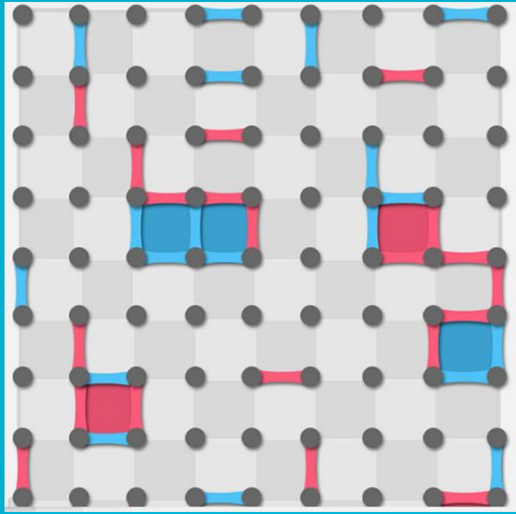
EE 566
Turn Based Strategy Game

Analysis Of Dots And Boxes

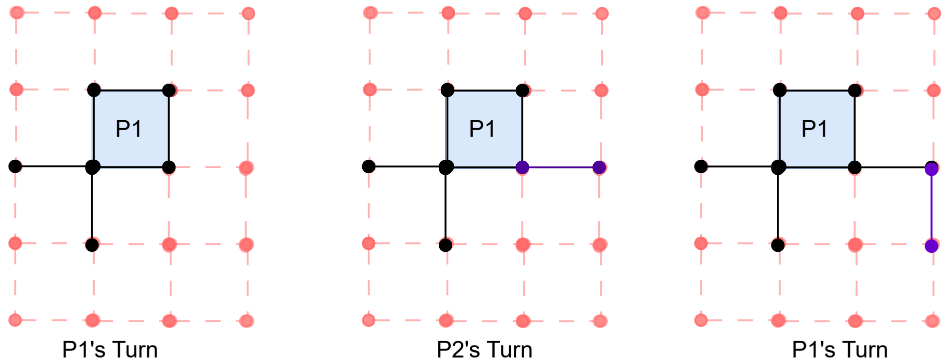


Group Members:
1) Shashwat Johri
2) Atharv Shendage
3) Ashmesh Dawande
4) Basant Solanky
5) Shivam Lodhi

Description Of The Game.

The Game Is Dots And Boxes , Which , I Am Sure , Everybody Might Have Played. The Rules Are:

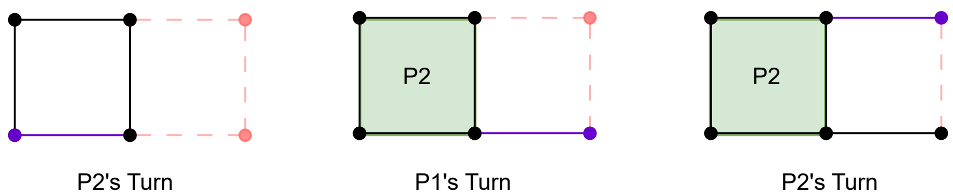
1. There is Grid Of Dots Of Some Size.
2. Every Player Takes Turn in Drawing An Edge
3. Last Player To Draw The Edge For A Square, Marks The Square For Oneself.
4. If You Complete A Square In Your Turn , You Must Draw Another Edge.



Simplifications For The Study.

To Study The Game As It Is Requires More Complex Analysis , As Far As This Report Is Concerned We Have Made Some Necessary Simplifications.

1. The Grid Is OF Size (2 x 3) i.e. 6 vertices.
2. Players Play Alternatively , Regardless Of If One Has Made A Box. .



Though P2 Made A Block, The Turns Changed Unlike Original Game

State Space.

Our State Space Contains List Of Structures Where Every Structure Is a List Containing The Edge Indices Of The Edges Which Are Part Of The State.

(Some Examples Are Shown For Explanation.)

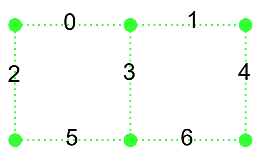
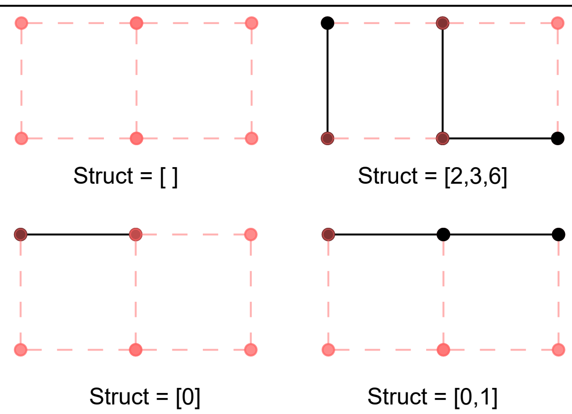


fig. Indices Of the Edges



```python
def draw(lst):
  # +-0-+-1-+
  # |   +   |
  # 2   3   4
  # |   +   |
  # +-5-+-6-+
  if type(lst) == int:
    if lst in range(0,48):
      lst = reps[lst]
    else:
      print("Cannot draw state num",
      return
```

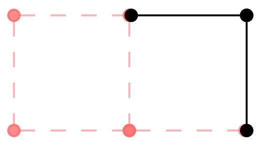
```
draw([0,1,2,3,4,5,6])

+---+---+
|   +   |
|   +   |
+---+---+
```

```
[ ]  draw([1,2,3,4,5,6])

+   +---+
|   +   |
|   +   |
+---+---+
```

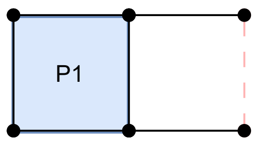
Corresposnding Code Snippet

State Space.

There Are Some States With Rotational Symmetry, Which For Our Purpose have The Same Structure , In Current Case There Are 128 Possible States And After Clubbing The Similar States, We have 48 States, So Now, The State is a tuple of It's Index In The State Space and P1's Score.

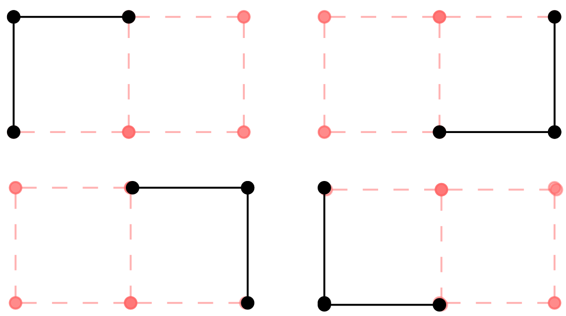


State=(4,0)



State=(45,1)

All These Structures Are Equivalent



State = (struc, p1_score)

```python
#state 0 2 4
state = [0,2,4]
draw(state)
print("After tranform 1 horizontal flip")
horiz = transform1(state)
draw(horiz)
print("After transform 2 vertical flip")
vert = transform2(state)
draw(vert)
```

```
+---+   +
|       |
|       |
+   +   +
After tranform 1
+   +---+
|       |
|       |
+   +   +
After transform 2
+   +   +
|       |
|       |
+---+   +
After transform 3
+   +   +
|       |
|       |
+   +---+
```

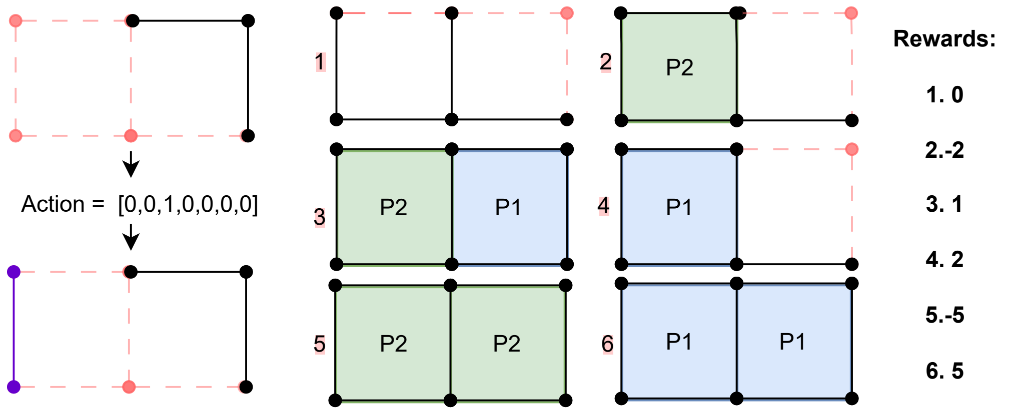
Corresposnding Code Snippet

Actions And Rewards.

A Action Is Basically Selecting Any Edge Which Can Be Thought As Selecting An Index From 0 to 6 , The Rewards Can Be Awarded Two Ways:

1. This Is Less Forgiving and Has Positive/Negative Reward Only At Terminal States, Rest are 0.

2. The Other Way Is To Keeping Track Of Scores At Each State And Giving Reward Accrodingly.



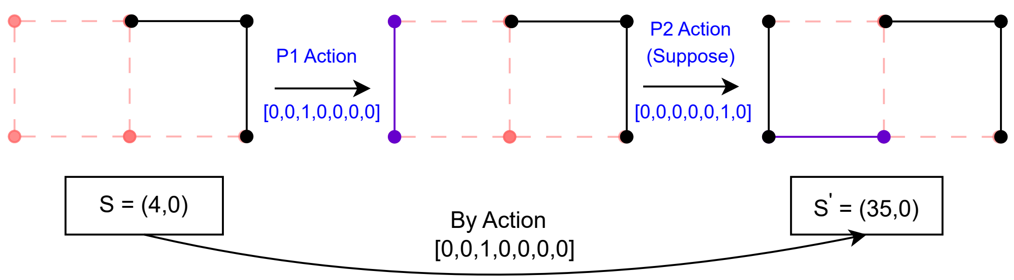

```
#defining rewards for states :
reward = dict()
for p1_score in [0,1,2]:
  for struc in range(0,48):
    box = boxes[struc]
    rew=0
    if box==1 and p1_score==1:
      rew=2
    if box==1 and p1_score==0:
      rew=-2

    if box==2 and p1_score==2:
      rew=5
    if box==2 and p1_score==1:
      rew=1
    if box==2 and p1_score==0:
      rew=-5
```

```
#defining rewards for states :
reward_2 = dict()
for p1_score in [0,1,2]:
  for struc in range(0,48):
    box = boxes[struc]
    rew=0
    if box==2 and p1_score==2:
      rew=5
    if box==2 and p1_score==1:
      rew=1
    if box==2 and p1_score==0:
      rew=-5
```

Corresposnding Code Snippet

Transition Probabilities.

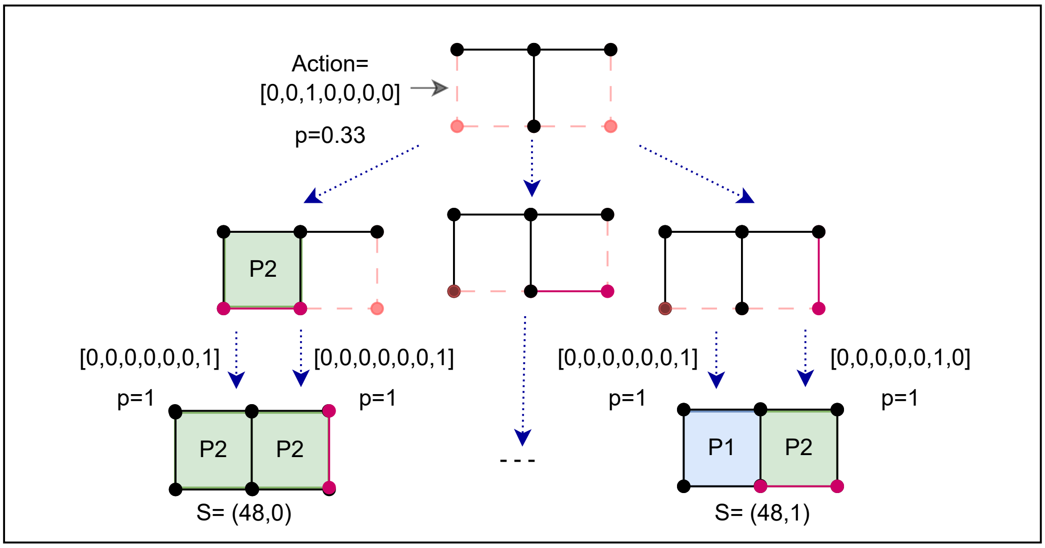
We Need To Look At States From Point Of View Of P1. Initially , We Have Taken Actions Of P2 As Randomly Picking Action From Available Actions And We Will Look At State When It Is Player 1's Turn Again And Not In Between.



T (S,A,S') = P = Probability Of P2 Choosing Edge 5 = 1/X = 1/4 =0.25

Transition Probabilities.

Some More Examples Of Transition Probabilities.




```python
actions = [0,1,2,3,4,5,6]
trans = [[[0 for i in range(48)] for i in range(48)]
#initializing transition probability matrix
#trans[action][start_state][end_state]

for state in reps:
  for action in actions:
    intermediate = next(state,action)
    if intermediate != None:
      #now orient the intermediate state to a rep st
      intermediate = state2rep[tuple(intermediate)]

 prob = 1/len(possible_states)

 for ps in possible_states:
   #same rep state can come in twice in possible_states,
   ps_num = rep2index[tuple(ps)]
   trans[action][state_num][ps_num] += prob
```

Corresposnding Code Snippet

```
+   +   +


+   +   +

 Player 1 Using
 action 0 on this
 state generates all
 states in the next
 turn of player 1

+---+---+


+   +   +

with probabilty  0.167

+---+   +
|
|
+   +   +

with probabilty  0.167

+   +---+
|
|
+   +   +

with probabilty  0.167
```

Value Iteration

The Bellman Expectation equation, giving the value of state '*s*' when following policy '*π*' is given by:

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

Equation 1: Bellman expectation equation giving the state value under policy π

where:

π(a|s) is the probability of taking action ***a*** in state ***s***.
p(s',r|s,a) is the probability of moving to the next state ***s'*** and getting reward ***r*** when starting in state **s** and taking action ***a***.
r is the reward received after taking this action.
γ is the discount factor.
v(s') is the value of the next state.

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

Equation 2: The Bellman expectation equation expressed as an update function.

Initialization:

We start with all the values of each state = 0
We initialize the policy to be stochastic (Random)
Discount factors 0.9 and 1 were tried(both gave same convergence)

Iteration:

Then we do a sweep through all the states and update the values of each state.
Multiple sweeps(iterations) are done until threshold condition is met

Threshold Condition:

Number of iterations > 300 or ,
Absolute difference between the previous and new value of state is less than 2

```
iter:  1
{(0, 0): 0.0, (1, 0): 0.0, (2, 0): 0.0, (3, 0):
iter:  2
{(0, 0): -0.11428571428571424, (1, 0): -0.333333
iter:  3
{(0, 0): -0.9142857142857146, (1, 0): -1.633333
iter:  4
```

…………

```
iter:  299
{(0, 0): 565.7142857142775, (1, 0): -386.433333333331,
iter:  300
{(0, 0): 567.6285714285631, (1, 0): -387.7333333333312,
iter:  301
{(0, 0): 569.5428571428487, (1, 0): -389.0333333333309,
```

Our Policy : Deterministic Mapping From State Space To Action Space.

```
+   +   +


+   +   +
Player 1 score: 0
[0, 0, 0, 1, 0, 0, 0]
+---+   +


+   +   +
Player 1 score: 0
[0, 0, 0, 1, 0, 0, 0]
+---+---+


+   +   +
Player 1 score: 0
[0, 0, 0, 1, 0, 0, 0]
+   +   +
|
|
+   +   +
Player 1 score: 0
[0, 0, 0, 1, 0, 0, 0]
+---+   +
|
|
+   +   +
Player 1 score: 0
[0, 0, 0, 0, 0, 1, 0]
+   +---+
```

Policy Improvement

Estimate[a|s] : Estimated value you're going to get after taking action a at state s
It can be calculated as :

$$E[a|s] = \sum_{s' \in S} p(s'|s,a)v(s')$$

Our initial stochastic policy is then improved using the values of each state.
For new policy
π'
For each state s, greedily choose the action that maximises the Estimated value at that state

$$\pi'(s|a) = 1, \quad argmax_a E[a|s]$$
$$\pi'(s|a) = 0, \quad o/w$$

Screenshot below shows one particular state and our policies response to it :

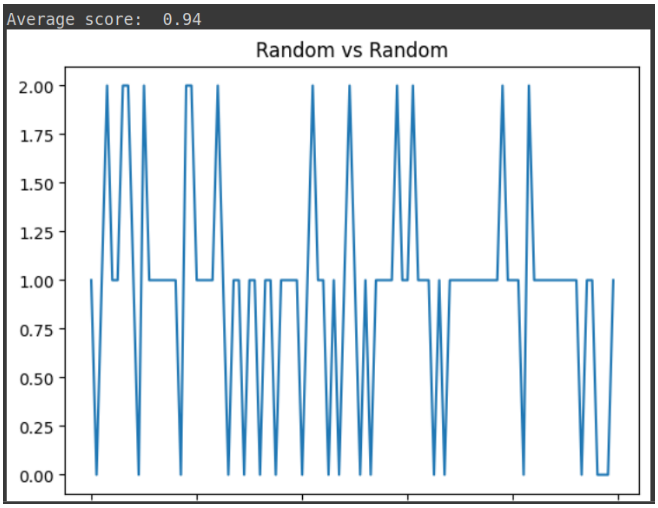
```
for act in legal_actions:
  estimate = 0
  p1_score_new = newscore(struc,act,p1_score)
  for final_struc,prob in enumerate(trans[act][struc]):
    if(prob==0):
      continue

    try : estimate+=prob*(new_value[(final_struc, p1_score_new)])
    except: print(final_struc, p1_score_new)
  if estimate > global_estimate:
    global_estimate = estimate
    greedy_act = act
```

Results

Simulating 100 matches between two stochastic(random) policies:

```
for i in range(100):
  score = simulate(policy_init,policy_init)
  scores.append(score)
```

Average score: 0.94



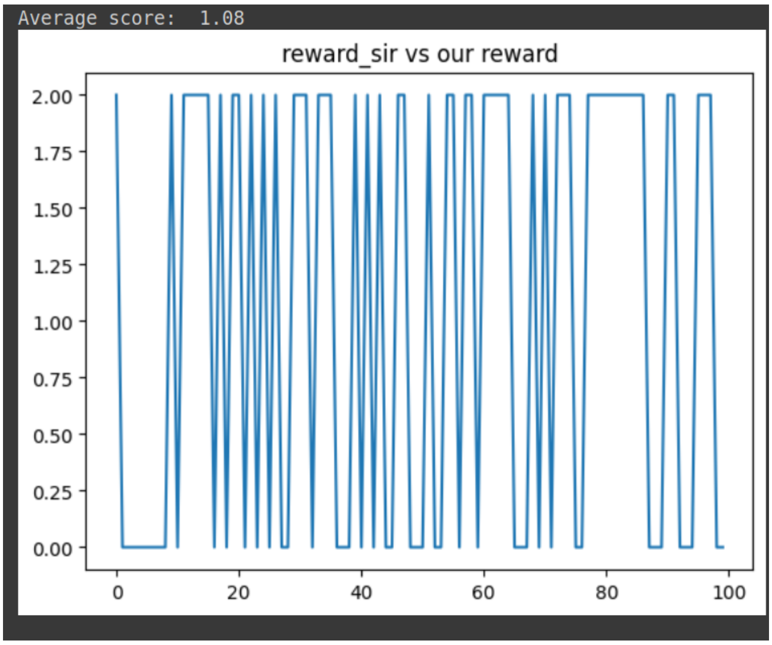
Simulating 100 matches between our improved policy vs random policy:

```
for i in range(100):
  score = simulate(new_policy, policy_init)
  scores.append(score)
  total+=score
```

Some other results

Fight between two reward systems:

Our initial rewards :
+2,-2 for intermediate boxes, +5,-5,0 for final game states
Sir's suggestion:
only +5,-5,0 for final game states



No ties! This means the player which starts is always winning.
values converge at the same values in both reward system so policy is effectively the same.