- Block: ⇒ A square of one colour
  ⇒ A square that is subdivided into 4 equal-sized blocks (children)
  - level: Number of divide needed to generate the block
    0, 1, 2 ...
  - ~~Maximum~~ level depth: 最深层数，不能无限延伸/recursion.
    最小单位称为 unit cell
  - 一个不被分割的 block 叫做 leaf.

Actions:

Swap horizontally: 左右互换 swap L/R children of a block
  vertically: swap top/bottom children of a block
- Smash: 将一个block分成 四个 random 颜色的 sub-block. 每一个 sub-block有一定机率被再次smash. 到 max-depth (unit cell)时无法被再次 smash
- Paint: 只能作用于 unit cell，将一个cell paint成该玩家的 target colour.
- Combine: 将一个 block 转化为 leaf，颜色由其 children 中占多数的颜色相同。若有多数存在，因个 children 组合成一个leaf. (无 tie)
  只能作用于一个block, which all 4 children are leaf (children 都没被分割)
- pass: do nothing.

操作：

· 选择block / level: hover mouse → 选择 block / level.

S → increase lv, W → decrease lv.

Goal & Scoring

目标及分 ① Blob goal: aim for ~~to~~ largest 'blob' given colour C.

连起来的最大区域，分数 = # of unit cells in that blob

② perimeter goal → 边长

aim to put ~~most~~ units of given colour C on perimeter.

普通边 +1分 角 +2分

unit cell

失分

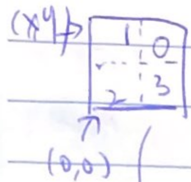
· Rotate, Swap, Pass cost 0分

· paint, combine cost 1分.

· Smash cost 2分.

· ~~Cas~~ Class Block

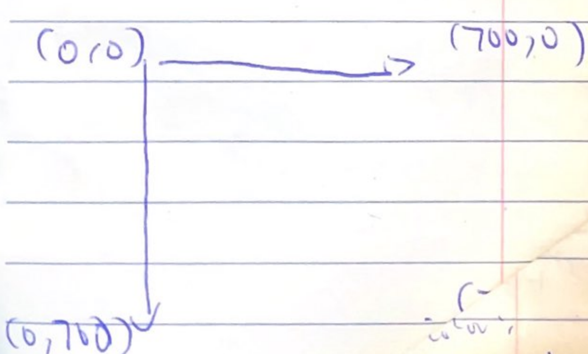
表示 tree-like structure，表示 block 所在的二维坐标。



· position: Block 的左上角坐标
· size: 单位 pixel
· colour: block 没被分，是 color；被分，None
· level: 当前 level
· max_dept: max level
· children [0, 1, 2, 3]

RI: 当前 level ≤ max level

· children 0个或4个
· 当 block 有 children { child max_depth 与 parent 相同
  child size 为 parent 一半
  ch. level 为 parent lv+1
  ... position 取决于第几个和 parent
  parent color 变为 None.
· 当 block 无 children: color 不为 None



append ( ).

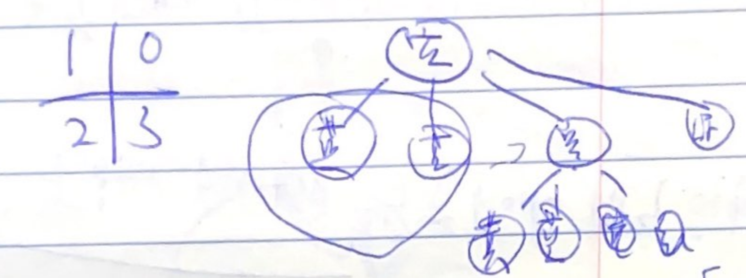
Return: list of tuple,

Task 1: Displaying Blocks

需 render board object into 图像.

这部分目的: make Block object drawable, by convert Block to a form that 能被绘制.

1. _block_to_squares: 注意, Block position, size, children



若 colour is None, res = [ ] Forall the children.

not

if not board.colour:

若无颜色: find all its children
go to all children

若有颜色:
~~colour it~~
append to it.

res = [ ] ← base case [colour, position, max_d

if 有 colour,
return [ ].

else: forall ... colour:
res.append( ).

list[tuple[tuple[int, int, int], tuple[int, int], int]

self.smashable():

可以 smash → return True

① generates 4 children ② parent.colour = None

for child in children:

RN = XX

if smash() & RN = XX:

self.smash()

else: 结束

不可以 smash → return False

children_position 是 block 的 children 的 position.

int(len(c_l) × random_num)

lv = max_d, block has children already

random_number — random number.

Task 2 Block smash. 关于 sub-block assign.

关于 block.py 中的 smash method → values.

smash 这个 method 的 return type 为 bool.

random_number — randomnum...

Task 2 Block smash.

写 block.py 中的 smash method → 给 sub_block assign values.

smash ~~中~~ method 的 return type 为 bool.

① call smashable → ~~True~~ False return False

~~False → return False~~

True → color = None
generate 4 child

For each child,
generate RN &
see if smash child

RN & smashable ~~True~~

else

Random ← ~~False~~

Assign number.

END

没到
check max_level 是 child 为 0. 不能分

not

if self.smashable():

if

→ 0.

~~...block~~

① return type : bool 代表smash行为是否成功

② 什么时候可以smash: self.smashable is true ✓

③ smash之后, 每个children 可能可以 smash.

若 (child.smashable ( )) & (Random Number 满足条件):

→ call smash on child

(若否) → 随机 Assign 颜色.

④ 若 self.(smashable ( ) is false) → do nothing ✓

Task 3. The goal classes & Random goals

I read goal.py 的 { Goal. { score / description
Blob Goal
Perimeter Goal

读

Task 3. The goal classes & Random goals

1. read goal.py 的 { Goal. { score, description
   读 | BlobGoal
   | PerimeterGoal

2. 写 goal.py 的 generate_goals
   BlobGoal descript. — 要用到 colour_name()
   PerimeterGoal description /

Goal 是 abstract ~~rat~~ class

> 0.5 per

• generate_goals ~~0 - 2 版~~ else.

① 每个 goal 可以是 perimeter，也可以是 Blob
② 所有 goal 颜色不同：怎么确定呢，create 颜色的的 copy.
   选一个，减一个

需要解决 ① 判断 position 在 (x,y) 在 block x 范围内

loop

- 若在，pass

若在，check lv — if lv == desire lv, return

if lv > desire lv, pass

if lv < desire lv

recursion, call function on all its child

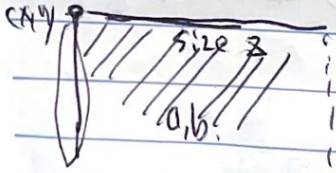
看 child — if no child → return 当前

if 有 child → loop & recursion

若找不到，return None

① Block: position (x,y), size,

point upper left, target (a,b)



$x \le a < x+z$

$y \le b < y+z$

location in block rage

x_coord ≤ location[0] < x_coord + width &, y_coord ≤ [1] < y_coord + ...

Task4: the player class & Random players

文件 player.py

最多支持 4 players.

1. 看 abstract class Player, < id, goal, penalty

2. Human player inherits Players class

≤ ... < x+z

y ≤ b < y+z

location in block range

x_coord ≤ location[0] < x_coord + width & y_coord ≤ [1] < y_coord+width

Task4: the player class & Random players

文件 player.py

最多支持 4 players

1. 看 abstract class Player. < id, goal, penalty

2. Human player inherits Players class

3. 写 function _get_block

4. 写 function create_players · 给每个player分配 id (从0开始), & Random goal.

player ① get_selected_block: return player 选中的 block.

② process_event 根据 pygame event 更新 player.

③ generate_move: 返回在 board 上可以操作的一个 move.

返回值是 tuple (Action, Block)

Human player. Attributes { _level: player 最近选中 block 的 lv; _desired_action: player 最近计划进行的 action }

get_selected_block: based on 当前 board, 鼠标位置, desire level

return 我们想操作的 block.

process_event. 是 event listener, 根据按键做操作。

例如 level up/down, 旋转...,

听到之后 update _desired_action 或是 _level

_get_block: input board, mouse_position, level

→ return 当前 block

· 若一个 block A 包括 position (x, y), A 的 ancestor 也包括, level 告诉 return 哪个 block.

若 input lv > 当前位置包括 lv 最深 block 的 lv, return deepest block.

· 若 no block found on pos, return None

block 包括内部, 上/左 edge.

create_players:

num_human → human player

num_random → random player

smart_player is a list; 每个 unit 是 smart_player 的 diff level

player id 从 0 到开始，每个 player 有 random goal.

generate_goal() 的 method. input int

return list of goals.

① 先 gen 全部goals，然后一个个 pn

num_players

goal_list =

Tasks

File: block.py

Task 5

File: block.py

①. 写 Block class 中余下 methods，每种都会 mutate Block

② 仔细读 Block 的 RI.

③ 写 Block._update_children_positions method，其它 method 会用到此 method.

提示：child block 对应的 coord (x,y)（即左上角坐标）不同

RI：① 当前 lv ≤ max_depth ② children 有 0 个或 4 个

③ 若当前 block x 有 children：children max_depth 和 x 相同，children size 是 x 一半，children 的 lv 是 x 的 level + 1，children 的 position 取决于 x 的 position，size 以及 children 是几号 child，x 的 color 为 None

④ 若当前 block 无 child：颜色不为 None

• _update_children_position input position

效果：把当然 Block 挪到 position new，update its child's position

return None

input：str 方向

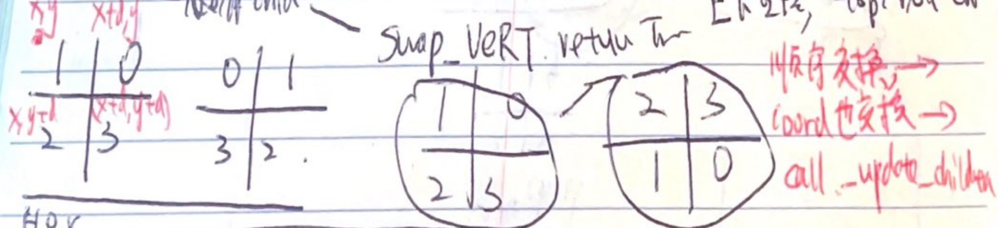
• swap return bool

无 child → return False

child

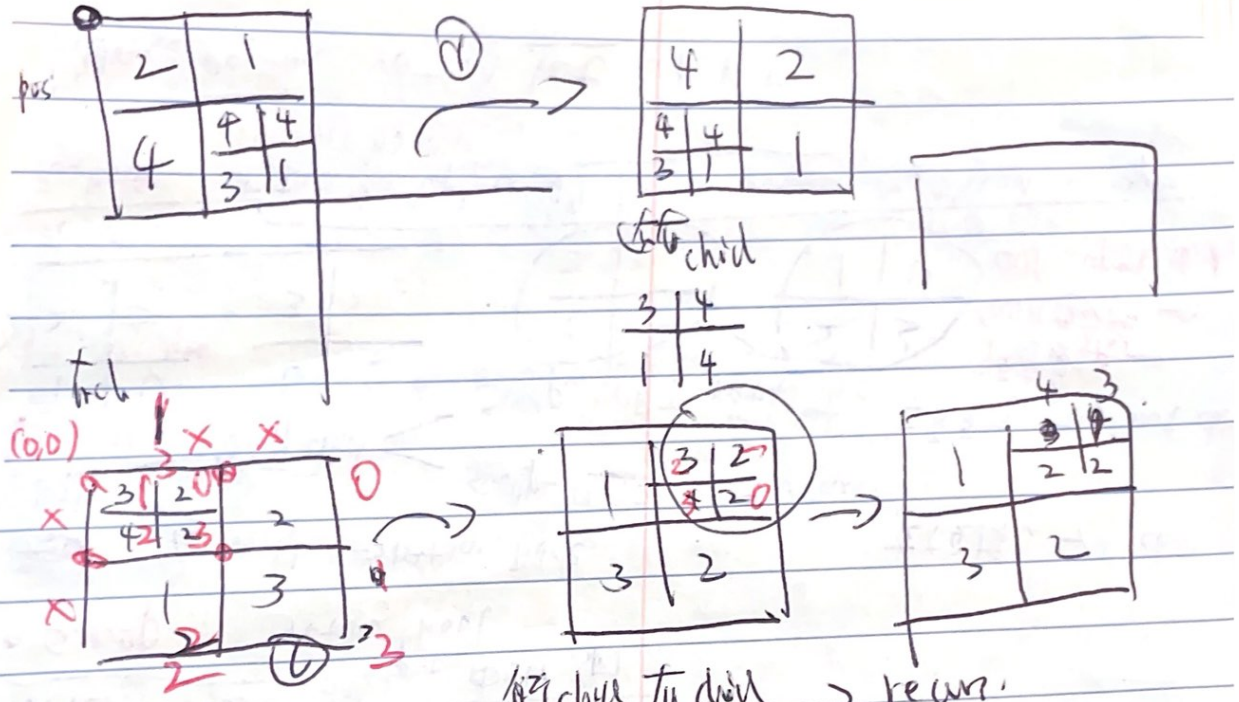
有 child → swap_HORZ → return True 左右互换，L/R child

swap_VERT return True 上下互换，top/bott child



Rotate：基本思路，update 自己 children position，然后用 recursion update children 的 chi

children_position return list of positi



child 0 的 index

child index 0

x, y =

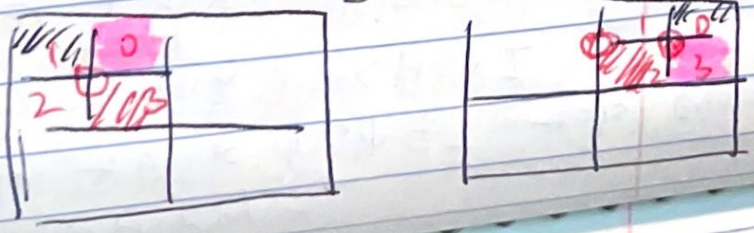
child. position

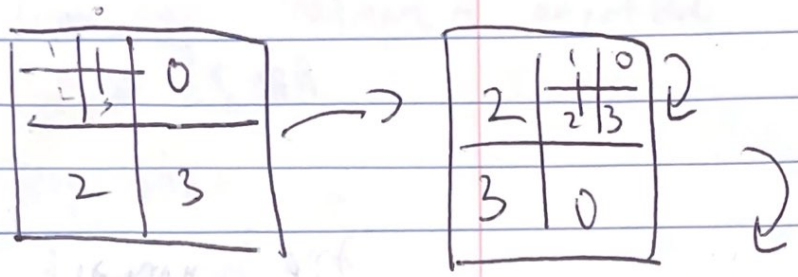
child. position = x, y+1.

1 = x+1, y.

2. = x, y-1.

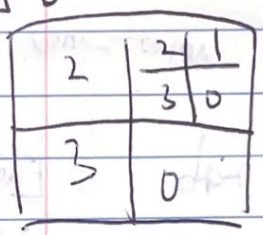
3 = x-1, y





Rotate a block 90 degrees

All children of block are also rotate 90 degrees



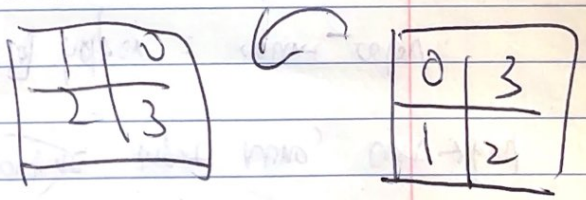
① 0,1,2,3 →

② record positions

③ 0,1,2,3 → 1,2,3 0.

* clockwise → ream.

CCW



paint input color, out → bool.

if lv = max_dept → F → return

T → change_c → return True

combine input None, out → bool.

helper : count_color.

{ } dict store, color & count

base case:

无 chil

return

.ch.

for chil in .ch:

dict[color] = .get + = (1)

→ recursion.

chid. count_color

return dict.

create_copy 深拷贝 input 不, output block

create_copy 两种input 无, output block

id不同, 元素相同

copy_block = 个.

if self. child :
    return copy_b

else: for chi in
        self chi. copy_build,

    copy_block. child. append

[[],[],[],[]]

Task6 : Perimeter goal

File: goal.py

① 看 PerimeterGoal class （最小计分单位是 unit cell，若大于 unit cell 要进行拆分）

② 写 flatten function

作用 将 Block obj 转化成 2D colour grid. Read doc string

③ 写 perimeter class 中的 score method （用 flatten）

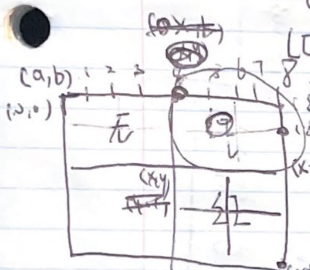
① PG 目标：最大化指定 colour 在 board 上的边长

② flatten ： input ： Block.

unit

Output： 2D list. 代表行，列，颜色.

L[i] 代表 column i, L[i][j] - unit cell at col i, row j.



(8,1) 一个 block 有颜色 → all the unit cell under the cell

(x+l,y) 是这个颜色.

block. position,

(x+l, y+l)   l = self.size.

无色 → call on child.

最细

unit_length = self.size / (2 ^ max_depth) → 一个 unit 的长度

判断当前的 leaf 占地多个 unit cell，在哪个范围

x_start_index = (x − a) / unit_length

x_end_index = (x + l − a) / unit length.

y_start = (y − b) / unit

y_end = (y + l − b) / unit

for x.

① base

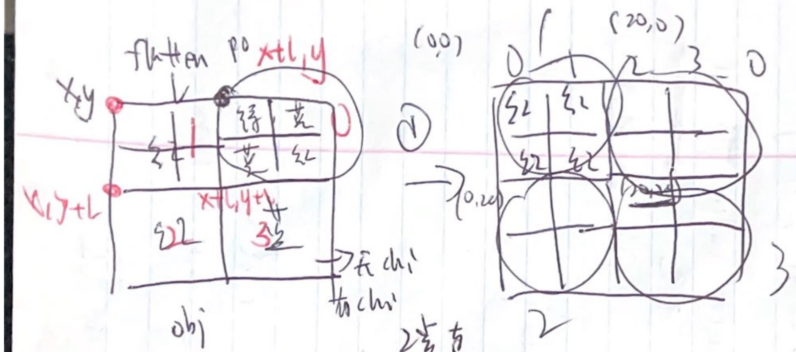
当前 block 有颜色 → 更新

通过该 block position, 覆盖范围 → index 范围

② 无颜色

→ check all children.

block: pos, size, cd, lv, max_d



→ 输出

2D list)

[[

block

→ 2D

2考虑

① 将非unit block 拆分为 unit block

所有 depth != max 且 有color 的 split

def split(block)

for child in block.children:

    if child.level != child.max_depth:

        if child.color: ← 有color ← has

            child.children = [

            child.color = None

        else: 无color

        split(child) ← recursion

].← 4个同color child

def (

• 当前是 unit

current_x, current_y = block.position

→ start_x, start_y

index_x, index_y

• 不是 unit

for c:

([ ], child, pos)

Task 7: Scoring for blob goals.

流程: flattening tree ⟶ iterating through cells in the flattened tree → find out 每个cell所属blob的大小，其中最大blob就是answer。
针对它颜色的

已知: 初始cell, target colour.
- 若颜色不是 target colour → cell size 0
- cell color = target color. → size ≥ 1. Ask the neighbour cells for their blob size (recursive), 加起来

潜在问题: 需要避免 double-count: 用 nested list 表示一个 cell 是否被 visit 过
-1 no, 0 visited != target colour, 1 visited = target colour

在 4×4 grid 中尝试模拟一下 recursion, 确认是否可行。
可以 assume 当 ≥ neighbour report answer 时可以得到正确答案
且正确 update visited.

① 看 BlobGoal _undiscovered_blob_size
input: · position, flatten 后的 2D board, 2D visited nested list
return: 在该 board 上的 position, 最大的含有 target_colour 的 blob 大小

- visited 最后每个 cell 都应是 0/1
- 若 position out of bound for <board>, return 0
(pos[0] > len(board)) or ([1] >)
blob_size = 0

① 判断自己颜色 match 且当前格子是 -1 (没有算过)
- 否 → update cell 0, return 0
- 是 → cell 1 blob_size +1, 去问上/下/左/右的 neigh

```
if (pos[0] >         ) (
    return 0.                                     base
else: # 当 还 not visited
    if visited [pos[0]] [pos[1]] == -1:
        if 颜色 match:
            blob_size = b_s +1 + re
        (不 match 不变)
```

if recu(上) + rec(下) + recu(左) + recu(右)
position 变

return

[[ ], [ ], [ ]]

① 绿 (0, 7) 5 → 10 → 14
蓝 (4, 6) 5 → 6

score

[[ ], [ ], [ ]],

- score

① 建 2D visited

② ~~loop~~ over board all
nested loop on all position of board,
find biggest blob size

③ return

Expected case:

Score should be updated for each type of goal.

player O : Olie Blob

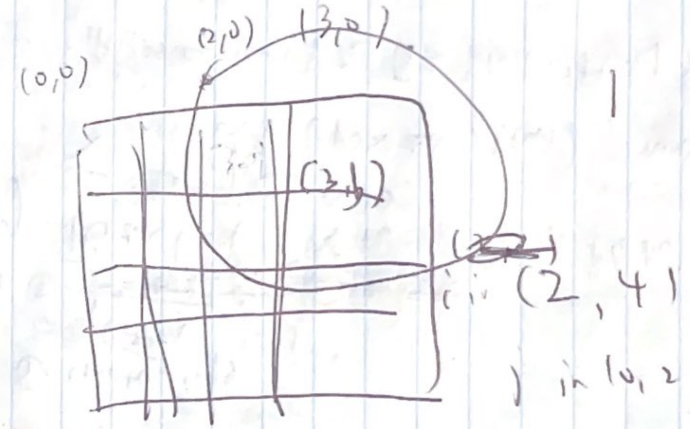
(4,0) 4 9

child is leaf

+2.

① (0,7) 5 → 10 → 14

(4,6) 5 → 6.



x_start = 2 (2,0) (2,1)
Y_start = 0 (3,0) (3,1)

x end = 2 + 2/1 = 4.

Y end = 0 + 4/2 = 2.

Task 8. Add random players

Files: player.py, block.py

· 理解 HumanPlayer：当按键，process_event() 会把 key press 转化为 desire action。当 action 不能被执行会 return None。之后 generate_move()，会根据 action 在 create 一个 move / 或 return None，这个 move 更新 board 然后 render 网格更新了出来

· ComputerPlayer class，是 RandomPlayer，SmartPlayer 的 parent。RP，SP 都会以相同与 HP 相似的方式与游戏互动。只是 process_event() 和 generate_move() 不同

ComputerPlay. process_events() 不需 listen to keyboard events

有一个 private bool attr：_proceed（一开始 False）。当 user click mouse during computer's turn → True → generate_move() 会 make a move → _proceed = False.

· 写 class RandomPlayer

让 RandomPlayer 在 board 上执行一个 valid move (generate_move)

1. .create_copy() 在 copy 上试

2. 在 copy 上执行 move，看是否成功

RandomPlayer.generate_move. → return (Action, Block)

在 player 回合时 return 一个可执行，随机的 move（不能 pass）

if not self._proceed:
    return None
else:
    copy = board.create_copy

copy →

Random move →

① smash, swap, rotate, paint, combine

② 位置

问题：在当前 block 上执行 Random_move 还是在当前 block / children（任意 block）（还是当前）

① create_copy

~~② flatten~~

② 随机 level，随机 choice，随机 move

— 不可行，redo

— 可行，记录 (position, level)，move block

③ 用 (position, level) 在原本 block 中找到该 block

(5,1)

本 → copy

① create_copy

① flatten 后

找左上角位置，找位置 level

本 → copy

① create_copy

② Find random block.

③ Find random Actio

④ 判断 block, action 能否执行 — 不行 → 重新 find random block

行 → (x, y) coordinate 坐标

通过 (position, level) 找到

在本block中找到小block,

已有一个 function — get_block

⑤ 找到本 block 的 block,

return (本block, Action).

0 - 9

x, y = copy. position

new_x, new_y = x + random int (size),

y + random int (size)

lv = (random (max_depth + 1))

get_block

2 Rotate clock_wise

5 R

swap ↕

swap (—)

Smash

combine

paint

random.randint(0, n)

0.1

助手 → ① flatten 后

找到这个顶点, 位置, level

问题 flatten 后只有 color 了

② 在 block

无 child — view [children 1, 2, 5]

有 child 1/level

max depth

1/10

1 - 1/level → children

recursion.

return

小 block

helper 1

③ 位置 move

无 child

有

小 block _cp. move

找到 block 中的小 block

— get_block

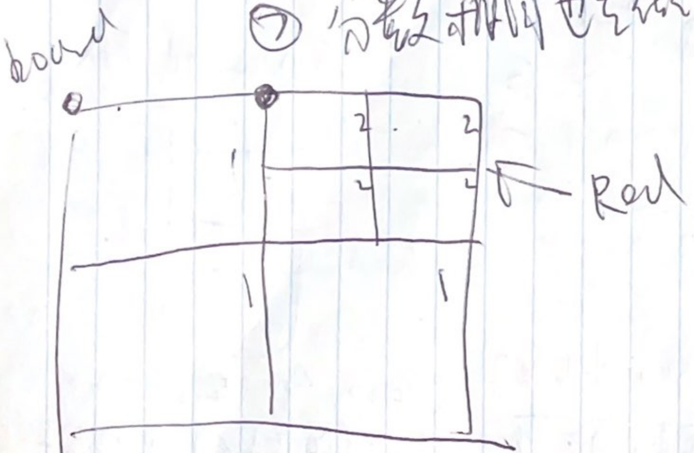
return (小 block, move).

~~random.shuffle~~

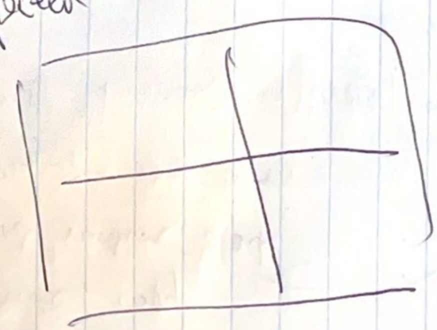
Rotation

两个问题 ① 很难 paint, combine, smash

② 分数排列也要做



expected



Snake 最大 黄色 跳5 → 9

最大blob 蓝 5 →

blob 2 8

黄色 ? → ? → ~

→ 7 +

P1 BGI 黄 → 10 → 18

Task9

Add smart players

File: player.py, Hack.py

class SmartPlayer

① 写 SmartPlaye.__init__

SP有diff lv.

② SP. generate_move.

生成n (n=diff lv) ~~move~~ valid move,

③ 找出其中分数最高的move(考虑 penalty)

记 create copy, 在算分数时 的move导致的

若无 best move, pass.

① logic fix : 只有加分才有 action

② penalty

1. CompPlay.__init__ ✓

2.

① 起始分数.

②

start_score

change_score, < action = None

- loop diff_lv 次:

每次随机一个 move.

算分, 更新. ← 减掉 penalty

- start_score = change

pass.