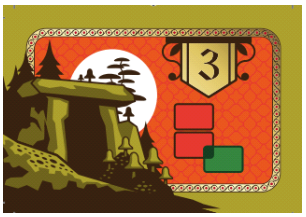
Points at the end of a game

Monday, September 18, 2023 2:19 PM

There are basically two kinds of configuration thanks to which you can get points at the end of a game:

1) groups of 3 diagonal pieces of the same colour

2) Two cards of the same colour one above another one with an angle of the two covered by another card of a different colour



Let's view the algorithms that I had in mind to evaluate the points given the game field:

1) GROUPS OF 3 CARDS OF THE SAME COLOUR

1.1 Function declaration

int Evaluate3cardsDiagonally(bool isSlopePositive, colour color_pattern)
isSlopePositive: the angle in which you are checking for the groups. it can be:
45 degrees (we can assign to it conventionally the value of TRUE) :

-45 degrees (we can assign to it conventionally the value of FALSE) :

then, we can use the method: keySet(): which returns in the form of a set all the coordinates(x,y)

in which there is a card in the game field.

1.2 Deciding the starting points (we start from left)

CASE IN WHICH ANGLE=45

By exploiting the property of the matrix we can divide the field in diagonal lines. for example, a possible diagonal would be the diagonal of the cards that have the coordinate that respects this equation: x-y=0. Making it a little bit more general we can make two ArrayListlist called **leftmost** and **rightmost** to keep track of the minimum coordinates and the max coordinates that respect the equation x-y=z. for each point we find when exploring the set of coordinates we check if the z was already present as a digonal in our ArrayList:

1) If yes:
   a. if x'< x we update that value of the ArrayList with the new leftmost point(x',y') ((x',y') are the coordinates of the new card we are evaluating while (x,y) the coordinate of the old minimum in leftmost)
   b. if x'> x we update that value of the ArrayList with the new rightmost point(x',y') ((x',y') are the coordinates of the new card we are evaluating while (x,y) the coordinate of the old max in rightmost)
   c. Else: we simply go on and check the other things.
2) if not we add the new x,y to the arraylist

PS: the 2 arraylist, since are compiled in the same moment, should have the reference for the same diagonal in the same index. eg: if the index 4 for the leftmost arraylist has let's say the point (-8,-8) also the index 4 of the rightmost arraylist should refer to the same diagonal → x-y= 0, an example could be (8,8).

CASE IN WHICH THE ANGLE=-45

It is the same but the rule for checking if it is the same diagonal is x+y=z

1.2 Evaluate the score

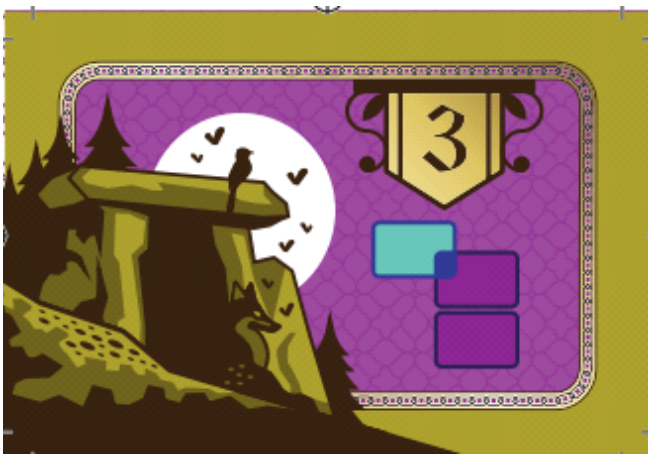
To count the score we need to start from the ArrayList of the starting point and follow the diagonal until we reach the point in the map that is in the rightmost part. To follow this path we have created a variable point called **direction** that simply at the end of the loop is added to a temp point that is used to go through the diagonal. direction will be equal to (2,2) in case isSlopePositive =true, false otherwise. Every time we find a card that has the same colour that we are checking for we add 1 of a counter named **CardsNum**. Every time we find a card of a different color (or we don't find the card, so there is a hole in the map) we reset that counter to 0. Whenever we add 1 to **CardsNum**, If **CardsNum**%3=0 && **CardsNum**>0 we add one to a counter named "**score**". We repeat this process for all the starting points of the ArrayList and we are done.

Note: every time we change the diagonal we need to reset the counter **NOC**

Time complexity: O(d*NC) where:
NC:=number of cards placed in the game field,
d:= number of diagonals

2) BLOCK OF CARDS

int EvaluateBlock(Angle angle, CardColor colourBlock, colour colourAngle)



angle: could be this time up right, up left, down right, down left.
colourBlock: represents the colour of the central block, in this case, purple.
colourAngle: represents the colour of the card in the angle, angle , in this case, blue.

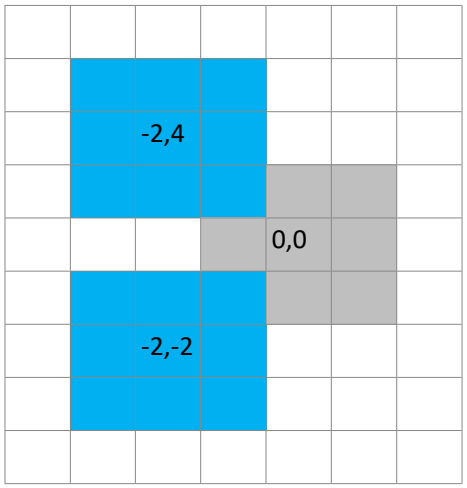
The strategy is more or less the same. As in the previous case we transform the map of the cards in a set. this time we create the arraylist of the cards that are in the lowest and toppest position for each column (x=c). If the config needed is like the purple above (or in general a config where the other card is on up left or up right angle) we would need to start from the bottom and go up until we reach the topmost position in that coloumn.

If instead we find ourselves in the situation where we have the other card in the bottom position we would need to descend from the top to the bottom.

As in the previous example, we need a **CardsNum** counter that is reinitialized to 0 each time we check another column and each time we see a colour different from colourBlock (or we can't detect a card but we have not reached the card in the top/bottom array) . We start checking each column starting from the cards that the arraylist is telling us to see and moving through the

column decrementing (or incrementing) **by 4** the y. If we see a card of colour colourBlock we add 1 to CardsNum iff CardsNum<1, if not it remains 2. After that if CardsNum=2 we check if the card at the corresponding angle is of the colour:colour_Angle if yes we update the final score and reinitialise the counter.

why is by 4 and not by 2?



as you can see in the short example we have some blank space in the middle that is responsible for that.

Time complexity: O(c*NC) where:
NC:=number of cards placed in the game field,
d:= number of coloumn