The Fruit Rage

Project description

In a typical zero sum two player game, players are generally competing for a certain common resource, and their gain is a function of their share of the resource. Often players have other challenges such as satisfying other constraints on other personal resources such as time, energy or computational power in the course of the game. Here we will introduce ***The Fruit Rage!*** a game that captures the nature of a zero sum two player game with strict limitation on allocated time for reasoning.

Your task is creating a software agent that can play this game against a human or another agent.

Rules of the game

The Fruit Rage is a two player game in which each player tries to maximize his/her share from a batch of fruits randomly placed in a box. The box is divided into cells and each cell is either empty or filled with one fruit of a specific type.

At the beginning of each game, all cells are filled with fruits. Players play in turn and can pick a cell of the box in their own turn and claim all fruit of the same type, in all cells that are connected to the selected cell through horizontal and vertical paths. For each selection or move the agent is rewarded a numeric value which is the square of the number of fruits claimed in that move. Once an agent picks the fruits from the cells, their empty place will be filled with other fruits on top of them (which fall down due to gravity), if any. In this game, no fruit is added during game play. Hence, players play until all fruits have been claimed.

Another big constraint of this game is that every agent has a limited amount of time to spend for thinking during the whole game. Spending more than the original allocated time will be penalized harshly. Each player is allocated a fixed total amount of time. When it is your turn to play, you will also be told how much remaining time you have. The time you take on each move will be subtracted from your total remaining time. If your remaining time reaches zero, your agent will automatically lose the game. Hence you should think about strategies for best use of your time (spend a lot of time on early moves, or on later moves?)

The overall score of each player is the sum of rewards gained for every turn. The game will terminate when there is no fruit left in the box or when a player has run out of time.

Game setup and examples

Figure 1 depicts a sample 10 x 10 game board with 4 types of fruits denoted by digits 0, 1, 2 and 3 in the cells. By analyzing the game, your agent should decide which location to pick next. Let's assume that it has decided to pick the cell highlighted in red and yellow in figure 1.

Figure 2 shows the result of executing this action: all the horizontally and vertically connected fruits of the same type (here, the selected fruit is of type 0) have been replaced by a * symbol (which represents an empty cell). The player will claim 14 fruits of type 0 because of this move and thus will be rewarded 14^2 = 196 points.

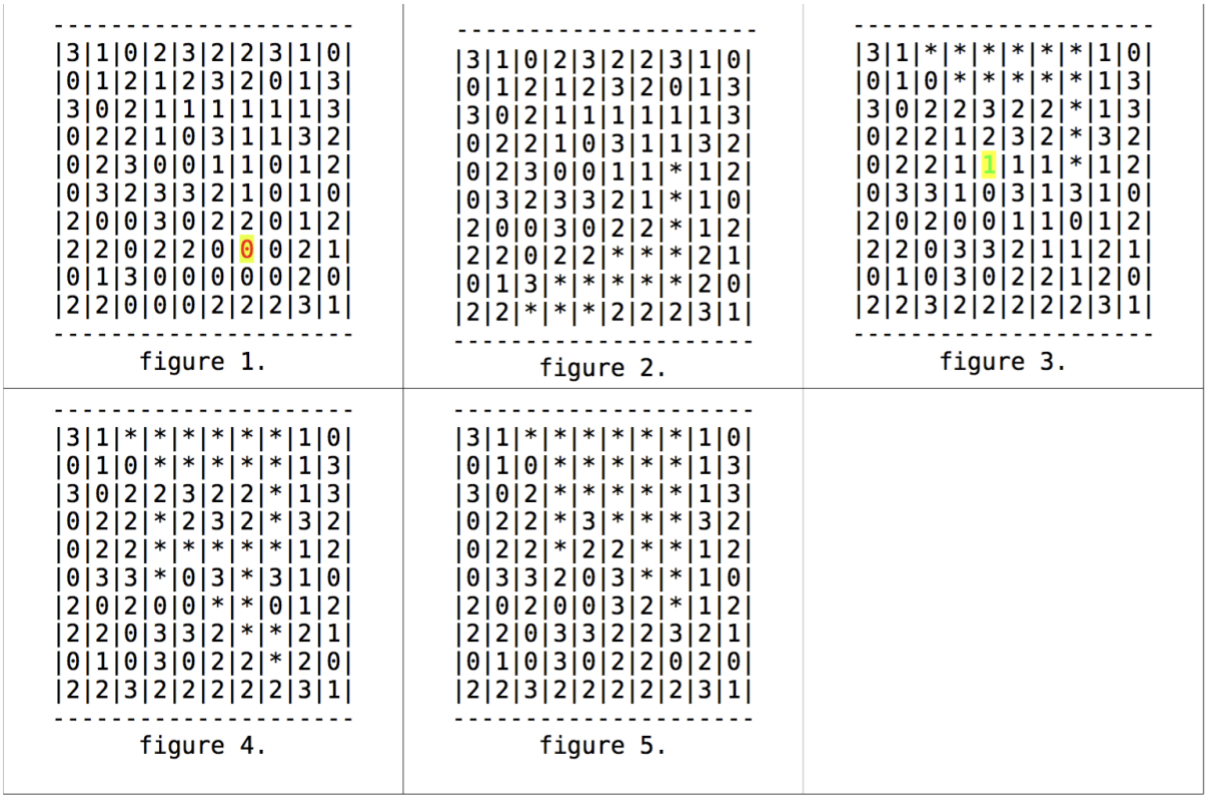

Figure 3 shows the state of the game after the empty space is filled with fruits falling from cells above. That is, for each cell with a * in figure 2, if fruits are present above, they will fall down. When a fruit that was on the top row falls down, its previous location is marked as empty (i.e., it becomes a * symbol). That is, no new fruits are injected to the top of the board. In addition to returning the column and row of your selected fruit, your agent will also need to return this resulting state after gravity has been applied. The game is over when all cells are empty, and the winner is determined by the total number of points, that is, sum of [fruits taken on each move]^2 (it is possible to end in a draw if both players score the same).

In figure 3, the opponent player then decided to pick the location highlighted in green and yellow. Upon selecting this cell, all 12 fruits of type 1 connected to that cell will be given to the opponent player and thus the opponent player will gain 12^2 = 144 points. In figure 4, cells connected to the selected cell are marked with * and in figure 5 you see how some of those picked fruits are replaced with the contents of cells above (fruits above fell down due to gravity).

To succeed, you should implement the **minimax algorithm with alpha-beta pruning**. While implementing minimax only (no pruning) might work in some cases, your agent is highly likely to run out of time for more complex cases unless you also implement alpha-beta pruning.

Input: The file `input.txt` in the current directory of your program will be formatted as follows:

First line: integer n, the width and height of the square board ($0 < n \leqq 26$)

Second line: integer p, the number of fruit types ($0 < p \leqq 9$)

Third line: strictly positive floating point number, your remaining time in seconds

Next n lines: the n x n board, with one board row per input file line, and n characters (plus end-of-line marker) on each line. Each character can be either a digit from 0 to p-1, or a * to denote an empty cell. Note: for ease of parsing, the extra horizontal and vertical lines shown in figures 1 – 5 will not be present in the actual input.txt (see below for examples).

Output: The file `output.txt` which your program creates in the current directory should be formatted as follows:

First line: your selected move, represented as two characters:
A letter from A to Z representing the column number (where A is the leftmost column, B is the next one to the right, etc), and
A number from 1 to 26 representing the row number (where 1 is the top row, 2 is the row below it, etc).

Next n lines: the n x n board just after your move and after gravity has been applied to make any fruits fall into holes created by your move taking away some fruits (like in figure 3).

Notes and hints

- Any additional heuristics on top of minimax + alpha-beta pruning which you can think of are allowed.
- The board you will be given as input could be completely filled with fruits, partially filled with fruits, but it will never be empty (all * symbols). Hence there will always be a valid possible move for your agent and you do not need to worry about passing your turn or detecting when the game is over.

- But note that the game board may become empty at some point during your search. You should then get the winner (or draw) by the number of points collected by each player.
- The board you will be given as input will always have gravity already applied (i.e., there cannot be a fruit above an empty space).
- Total play time will be 5 minutes (300.0 seconds).
- Play time used on each move is the total combined CPU time as measured by the Unix **`time`** command. This command measures pure computation time used by your agent, and discards time taken by the operating system, disk I/O, program loading, etc. Beware that it cumulates time spent in any threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as using 40 seconds of allocated time).
- If your agent runs for more than its given play time (in input.txt) + 10 seconds (grace period), it will be killed and will lose the test case or the game (see below).
- The grace period is only so that we do not kill your agent prematurely, and you should not count on it. You should aim to write out your output before your allocated time has passed. The actual play time taken by your agent will be considered and you will lose if it exceeds your allocated play time.
- You need to think and strategize how to best use your allocated time. In particular, you need to decide on how deep to carry your search, on each move. In some cases, your agent might be given only a very short amount of time (e.g., 5 seconds), for example towards the end of a competition run (see below). Your agent should be prepared for that and return a quick decision to avoid losing by running over time.
- You need to think hard about how to design your eval function (which gives a value to a board when it is not game over yet).

Example 1:

For this input.txt:

```
2
3
123.6
01
21
```

one possible correct output.txt is:

```
B1
0*
2*
```

(remember: B1 means second column, top row, i.e., the agent picked the top right corner cell and got 2 fruits of type 1. It received 2^2 = 4 points for that move).

Example 2:

For this input.txt:

```
3
1
257.2
***
***
*0*
```

one possible correct output.txt is:

```
B3
***
***
***
```

Example 3:

For this input.txt:

```
3
2
24.345
***
*10
000
```

one possible correct output.txt is:

```
C2
***
***
*1*
```

Example 4:

For this input.txt:

```
3
10
7.24
444
444
```

```
444
```

one possible correct output.txt is:

```
A1
***
***
***
```

Example 5:

For this input.txt (same layout as in Figure 1):

```
10
4
1.276
3102322310
0121232013
3021111113
0221031132
0230011012
0323321010
2003022012
2202200021
0130000020
2200022231
```

one possible correct output.txt is (same move as in Figure 1, and same resulting layout as in Figure 3):

```
G8
31******10
010*****13
3022322*13
0221232*32
0221111*12
0331031310
2020011012
2203321121
0103022120
2232222231
```