**Project description**

***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**, as studied

in class. 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.