General Plan

Balancer

Final Project for CS-C2120

Hau Phan
886690

Bachelor Degree in Science
Data Science Program
Years of study: 2020-2023

Department of Computer Science
Aalto University
Finland, February 26, 2021

Contents

1 General Description

A balance game where one tries to stack as many weights as possible to get the most score. Player with the most score win the round. Player win the most round win the entire game.

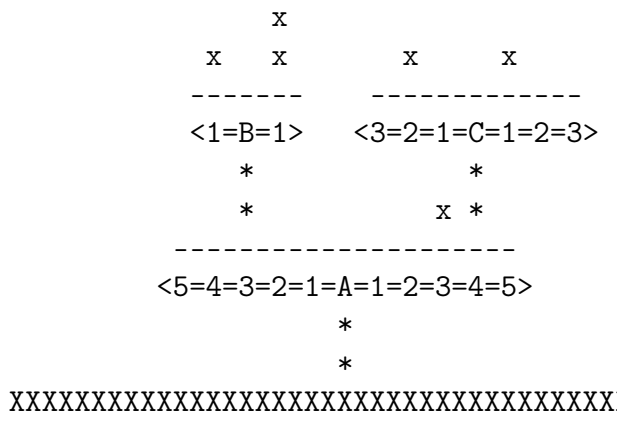
1.1 Important Concepts

1.1.1 Weight

A simple weight that have some mass. All the weights in the game have the same mass of 1. Each weight yields more score the further they are from the center of the scale.

1.1.2 Scale

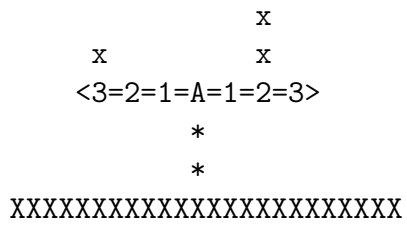
A scale means a supported long arm on which you can place **other scales** and **weights**:



More torque is applied to the arm if the weights are further from the middle point. A small imbalance is permissible. The maximum allowed magnitude of the imbalance is the same as if 1 weight was placed at the end of the scale before the other weights were set. i.e. imbalance $\geq$ the "radius" of the scale. (See the example below).

Once a scale is flipped (becomes imbalance), it will be lost forever along with the weights on it.

Example 1



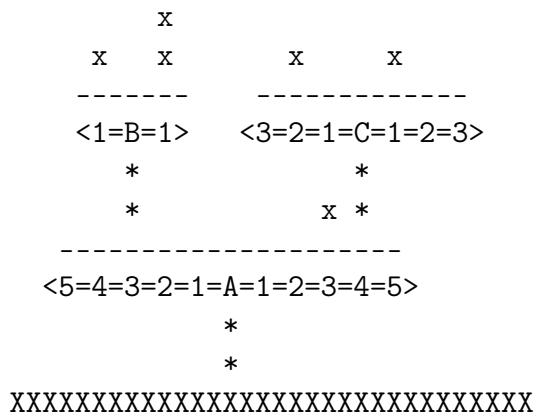
If a weight is at a distance of 3 from the center of the arm, it pushes that side down with the force of three weights. If the other side has two weights at a distance of two, they push that side down with a force of four weights. (2 * 2 = 4) The imbalance is 3 - 4 = -1. The absolute value of the imbalance is less than the radius of the scale, 3, so the scale is balanced.

Example 2

If the left side of the previous example did not have a weight, the scale would have been unbalanced (flipped) since the imbalance would have been 4, which exceeds the radius of the scale. In this situation, the scale is flipped and all the weights is lost.

You can also place other scales on the scale. The scale placed on the second scale applies a force equal to the sum of the weights and scales on it.

Example 3



Scale B is balanced, the right-side weighs 2 * 1 and the left one 1 * 1 which does not exceed the radius of the scale. The scale B itself weighs three weights.

Scale C is balanced since the left side weighs 1 * 2 = 2 and the right side 1 * 1 = 1. The difference of these is 1, which does not exceed the radius of the balance. Scale C weighs 2 weights!

Scale A is also balanced. On the left side there is a scale B at a distance of 3, 3 * 3 = 9. On the right there is scale C at a distance of 4, 2 * 4 = 8, and a weight

at a distance 3. All together there is a weight of 11 weights on the right side. The difference is 11 - 9 = 2, which is smaller than the radius of A, so the scale is balanced.

1.1.3 Players

Players can be either **human** or **bot**. There can be many players in a game/round (Usually 2-3 players).

1.1.4 Capture and ownership

Both the **weights** and **scales** during each round is either **player-owned** or **wild** which are owned by no one. The wild weights and scales are placed randomly at the beginning of each round and can be captured by players. Each players can also captures each other weights and scales. Captured scales give addition "buffs" for the owner's weights on the scale.

Weights can be **stacked**. One can capture all the weights by placing his/her weights on top of the stack.

To capture a scale, one must have **at least** r weights more compared to the player with the 2nd most weights on the scale, where r is the radius of the scale. (See the example below)

Example:

```
          a
 a b      a c
<2=1=A=1=2>
      1
      *
      *
XXXXXXXXXXXXXXXXXXXX
```

Here the scale is captured by player "a", since there are 3 "a" weights and only 1 "b" and "c" weight. The difference is $3 - 1 = 2 \geq 2$ the radius of the scale.

In the beginning, all scales are wild and available to capture. Owner of a scale will have the following benefits:

- His/her weights on the scale can not be captured, even when other players place the weight on top. The moment the scales' owner change, all weight that should be captured will be updated according to the rules.
- Each of his/her weight will have its score multiplied by 2.

1.2 Rules and Gameplay

1.2.1 Gameplay

- There is a predefined number of round in one game. (Usually 5 for 2-3 players)
- Each round will have a fixed number of weights that all players will draw from and place onto the scales. (Usually 8-12 weights per round)
- At the beginning of each round, one scale and 2-3 wild weights will be randomly placed by the computer with some predefined probability.
- The round starts and the players place the weights one at a time on the scales. The first player to place is the winner of last round. It will be chosen at random on the first round.
- The round is over when there is no weight left. The score is then calculated and the player with the highest score win the round.
- The next round then starts and **continues** from the last round's state.
- The player who win the most round win the game.

1.2.2 Scoring

Example:

```
        a
 a b    a
<2=1=A=1=2>
     2
     *          c b
     *   ?      c b
    <3=2=1=?=1=2=3>
           1
           *
           *
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Status: equalibrium

a: scale 2: (2 * 1 + 1 * 2) * 2 = 8 points
   scale 1: 3 * 8 = 24 points
   ------------------------------
   total: 24 points
```

```
b: scale 2: 1 * 1 = 1 points
   scale 1: 3 * 1 + 3 * 2 = 9 points
   ------------------------------
   total: 9 points

c: scale 1: 2 * 2 = 4 points
   ------------------------------
   total: 4 points
```

Here "?" represents wild weights and uncaptured scales.

The scale with index 2 is capture by the player "a" so its points is multiplied by 2

1.3 Difficulty level

Targeted level of difficulty: **Intermediate/Difficult**

The project first aims to meet all the requirement for Easy and Intermediate, which require a text-based user interface and a graphical one. To meet the Difficult requirements, the project also implements an intelligent computer opponents and extends existing rules by introducing the capturing and wild weights mechanics.

Additional features: **Online Multiplayer/RL Bot**

These are the two additional features that I am personally really interested in implementing and learning more about. However, it is not guaranteed that these functionality will be fully realized at the final deadline.

2 User Interface

2.1 Text based

In each turn, the program will print out the current state of the game in a format similar to the example format below. The program will print out the round number, the current score of each players and the state of each scale. Then it will ask for user inputs and place the weight accordingly. An example output would be as follow:

```
#################### ROUND 1 ####################
_________________________________________________
| Score board                                   |
|_______________________________________________|
| a: 27 points                           human  |
| b: 18 points                           bot    |
| c: 15 points                           bot    |
|_______________________________________________|
```

```
Game State:

Scale [a]
------------------------------
<a, 6>: <<d,3>--[a]<b,1>-|b,b|>
TORQUE: [9--3--7]
STATUS: Balanced
OWNER : None


Scale [b]
------------------------------
<b, 1>: <--[b]-<c,1>>
TORQUE: [0--2--2]
STATUS: Balanced
OWNER : None


Scale [c]
------------------------------
<c, 1>: <|b|---[c]---->
TORQUE: [4--4--0]
STATUS: Balanced
OWNER : None


Scale [d]
------------------------------
<d, 3>: <|c,c|---[d]---|a|>
WEIGHT: [8--4--4]
STATUS: Balanced
OWNER : None

>>>>>>>>>>>>>>>>> A(HUMAN) TURN <<<<<<<<<<<<<<<<<

Which scale ? (a,b,c,d): 3
Which arm ? (left/right): right
How far from the center ? (1-4): 4
```

In the example output, the scoreboard and the user input section is quite straightforward. The game state is just a listing of each scale's state, in the order of the index. For example, consider the scale with index 1, which is also the scale being placed on the ground:

```
Scale [a]
------------------------------
<a, 6>: <<d,3>--[a]<b,1>-|b,b|>
TORQUE: [9--3--7]
STATUS: Balanced
OWNER : None
```

Here <a,6> is its code ("a") and its weight ("6"). A scale is thus represented in the form:

$$< \text{scale's code}, \text{scale's weight} >$$

On the parent scale, child scales are also represented is this way: <d,3> is scale "d" with 3 weight on it, <b,1> is scale "b" with 1 weight on it.

[*scale'scode*] i.e [a], [b],... represents the center of the scale and each hyphen (-) denotes an empty space on the scale with no weights.

A stack of weights is represented by two vertical bars surrounding the weights. Each weight is represented by its respective owners' letter: i.e "a", "b", "c", ... The bottom of the stack is equivalent to the left vertical bar and the top of stack corresponds to the right vertical bar.

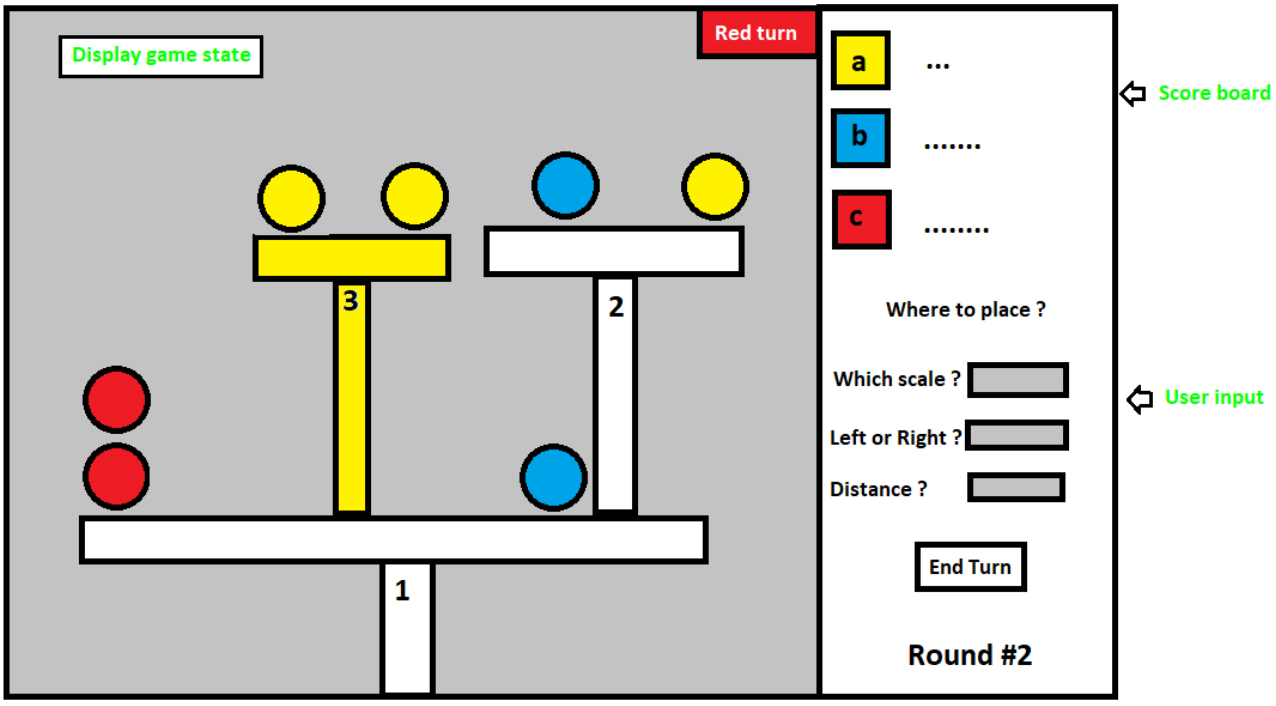
"TORQUE:" contains three numbers (ex: [9–6–7]), from left to right are: the left torque, the radius of the scale and the right torque. If the difference between the left and right number is greater than the middle number, the scale is imbalance.

"STATUS:" and "OWNER:" indicate whether the scale is balance or not and the owner of the scale respectively (see subsubsection 1.1.4)

2.2 Graphical User Interface

The user can also interact with the program through a graphical user interface. There are three main components in the main UI: the game screen, the scoreboard and the form.

- Game screen: display the current state of the game which includes where the scales are located, how the weights are placed, who currently owns which scales (by color),...
- Scoreboard: display all the relevant information for the user to make a decision on where to place his/her weight, which might include the current score, the number of weights and scales of each players, the number of weights left...
- Form: where the users can fill in the information of where the weight should be placed. There is also a submit button to confirm the placement of the weight.



3 Files and file formats

The state of the game can be saved in semi-readable text format as follow:

```
BALANCER 1.3 SAVE FILE

# Meta
Number of Player: 3
Human: a
Bot: b,c
Round: 2
Turn: c

# Scale
_,_,5   : -4,c,c | 2,b
a,3,4   : -1,b |   2,a,c
a,-1,4  : 1,a    |-1,a

END
```

- The first line of the file must be of the form:

  BALANCER (VERSION) SAVE FILE

- Each block of data has a header and a body. The first character of the header must be the "#" symbol to be identified as a header. The header marks the start of the block.

- Each row in the body contains information about the state of the game and is of the form:

  IDENTIFIER : DATA

- The IDENTIFIER and DATA is different for different block, here there are only two different type of block: **Meta** and **Scale**.

- The Meta block contains metadata of the game. There are 6 main identifier in the Meta block, which is shown in the example above. More identifier might be added in the course of development.

- The Scale block contains the position of each scales and weights when the game is saved. The IDENTIFIER is the scale's position and the DATA is the position of each weight on the scale:

- Noted: In the following example file, negative indices indicate the object is on the left arm of the whichever scale it is on, which we will call the **parent scale**. Positive indices indicate the object is on the right arm.

- For example, this line in the Scale block:

  ```
  1,3,3  : -1,b  |  2,a,c
  ```

  from left to right indicates: the parent scale (scale with index 1), where on the parent scale this scale is placed (on the right arm, distance 3 from the center) and the radius (the scale have radius 3)

- After the colon are the weights' position on the scale, it must be listed from left to right (thus the indices must be from smallest to biggest).

  - (-1,b) indicates on the left arm, distance 1 from the center, there is a weight belong to the b player
  - (2,a,c) indicates on the right arm, distance 2 from the center, there are weights stacked on each other. The bottom weight belong to player a and the top weight belong to player c.

- The END keyword represents the end of the saved file. All line after this keyword will not be interpreted by the parser.

- Noted: Any excess white space will be ignored by the parser.