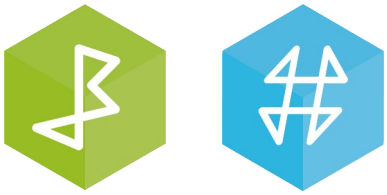
Chu Thomas
Guillou Aurélien

IUT de Belfort
Département Informatique
S1-B1
Groupe numéro 3

Comparison study of the missing token game's AI



AI algorithm description

We created three AI for the missing token game:

- an AI(0) playing at a random location inside any empty space available
- an AI(1) playing “aggressively”, it plays a token near the last opponent's token played, giving their token of “little weight”, a less likely chance of being near the last empty space
- an AI(2) playing “defensively”, it plays a token near their own tokens, the goal is to fill the most space so that the last empty space cannot be surrounded by this AI’s color

```
// ia naïve random
if (choix == 0)
    joue = (int) (Math.random() * NCASES);
```

The data structure and technique of the random AI is fairly simple, the “joue” variable takes a random value from 0 to the number of squares (NCASES), doing so as long as an empty space isn’t chosen thanks to the do-while’s condition in the main method. This algorithm has a constant time complexity.

```
else if (choix == 1) {
    if (bleu.size()==0) {
        joue = (vide.get((int)(Math.random()*vide.size())));
    }
    for(int li = 0; li<bleu.size(); li++)
    {
        getVoisins(tNear, bleu.get(li));
        for(int lj = 0; lj <tNear.size(); lj++)
            voisinsColor.add(tNear.get(lj));
        tNear.clear();
    }

    voisinsColor = rmDoublon(voisinsColor);
    Collections.sort(voisinsColor);

    for (int i = 0; i < vide.size(); i++) {
        for (int j = 0; j < voisinsColor.size(); j++) {
            if(voisinsColor.get(j) == (vide.get(i))){
                joue = voisinsColor.get(j);
            }
        }
    }
```

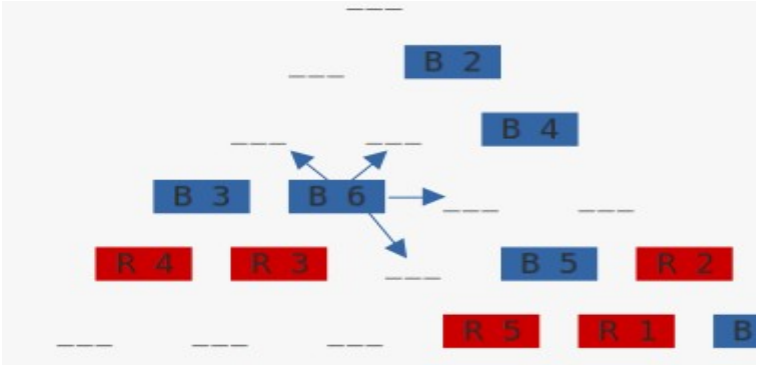
Things are getting more interesting with the aggressive AI(1) and defensive one (they basically work the same), first we made some list to contain the placement of everything, empty spaces, red

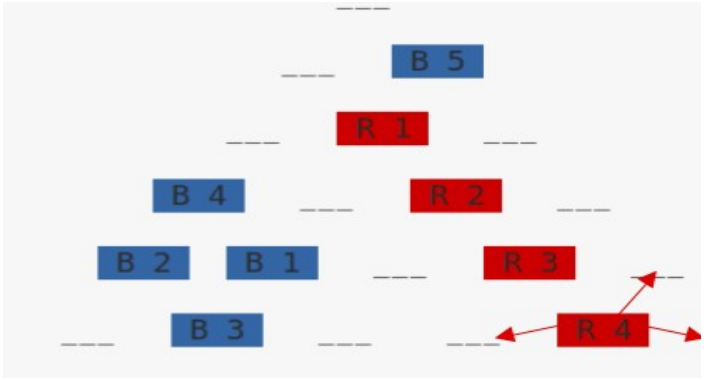
squares and blue squares, those list are filled every turn simply because their values changes each turn. For both AI(1) and AI(2), their strategy implies that they have played at least 1 time, meaning1 we need a special case for the turn 0. After that we fill the voisinsColor list with a method getting a list of neighbors of a given id and using that method to get every single neighbors of a given color. We then delete duplicate values using an Hashset method and sort the list. Essentially what we want is to know if there is an empty space near the enemies' last token. To do that we make the last to for loops and give the "joue" variable any id that's available in the "vide" list.

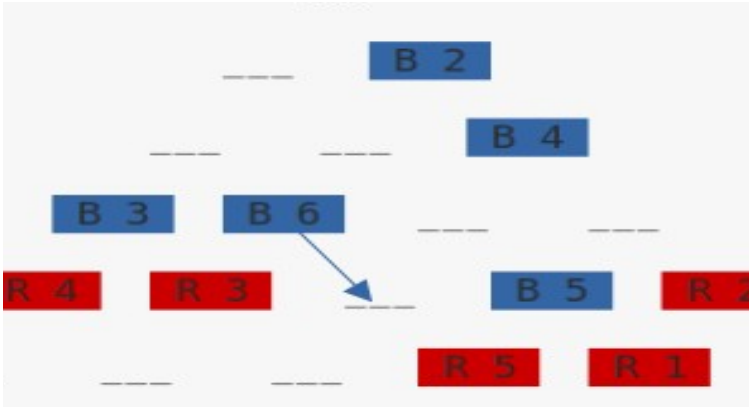
Here's some pictures illustrating their process:

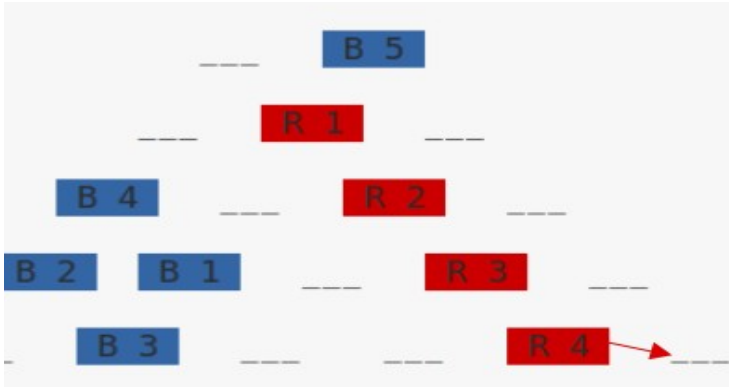
IA(1) « agressive

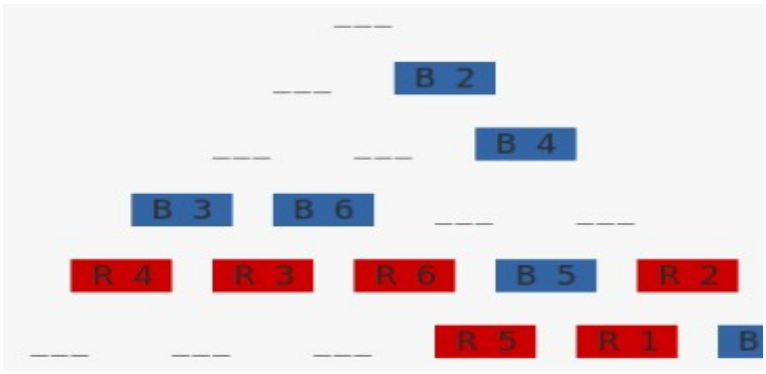
IA(2) defensive

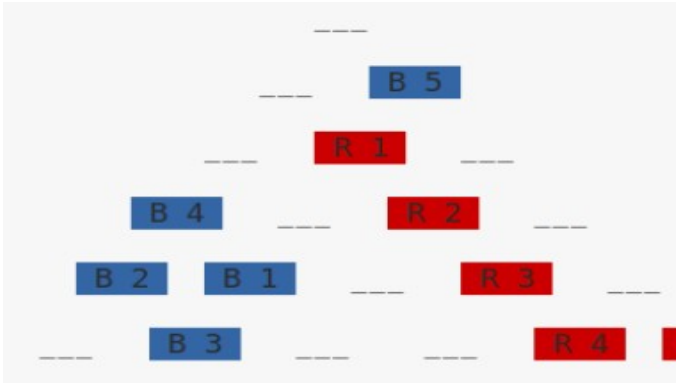










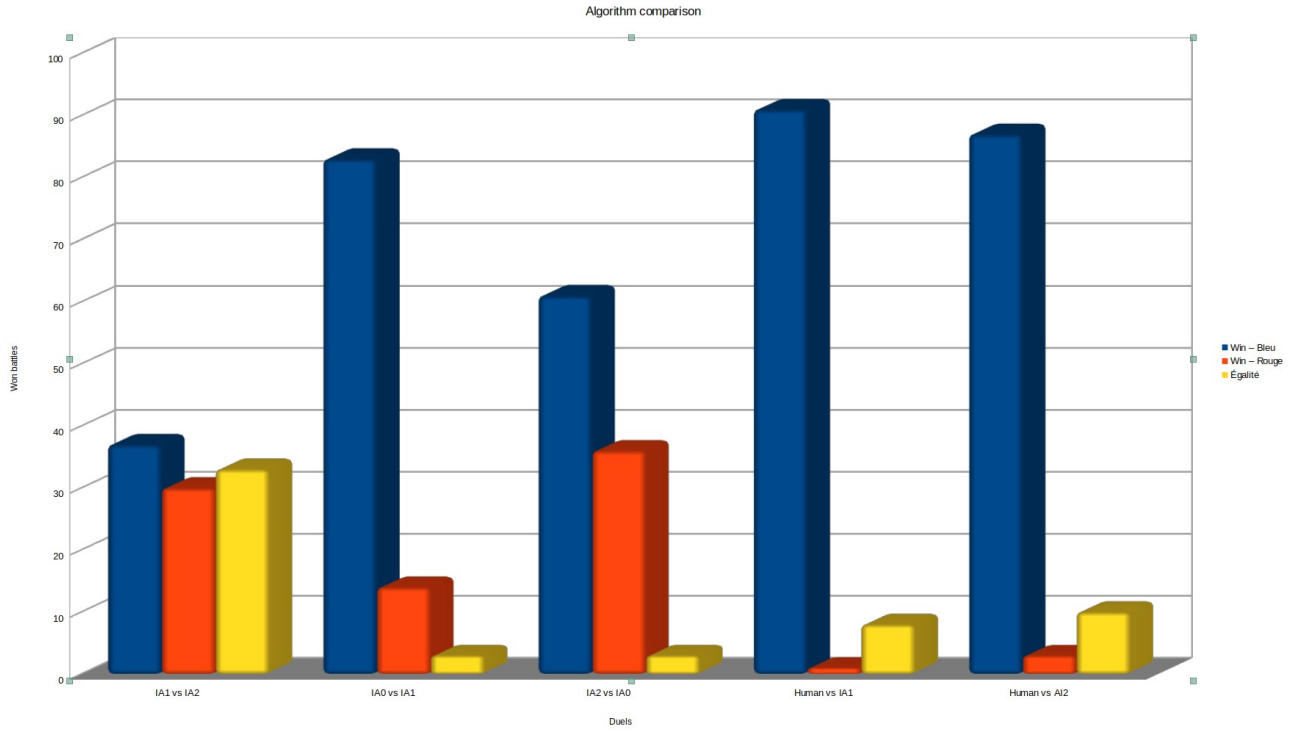


Comparison

We will now compare the performances between the aggressive AI(1) and the defensive AI(2):

- We tried to mesure elapsed time, calculate execution time. Using the Profiler class, we took a time before the AI algorithm and a second one after, subtracted the two and obtained an average execution time of 0.036379ms elapsed time for the AI(1) and 0.0248225ms elapsed time for the AI(2), leaving a difference of 0.01ms on average. Not a surprising conclusion considering that the two have pretty much the same structure.
- Similarly, the memory used for both AI are one and the same, around 31,195 Mo.

Then we decided to make a little statistical study on both AI's ability to win against a variety of opponents, giving us this histogram:



We can see that against each other, the defensive and aggressive AI seem to have an equal amount of winrate, there's no decisive winner. Although the aggressive AI does a lot better against the random AI than it's defensive counterpart who still has an advantage but seems to struggle a bit more. Sadly, it looks like it's a no match against a real human, our AI can't adapt like a human. Both AI have a similar time complexity and memory usage, but depending on their opponent, they do perform a bit differently.

Conclusion

Ultimately, the offensive AI seems to do a little bit better compared to the defensive AI. They also both beat the random AI, meaning they’re still better than a “naive” AI. A thing to know is that our program isn’t really “clean” , a bug seems to occur sometimes leading to an endless loop. We tried to find the combination causing that problem but we weren’t able to fix it.