Minesweeper solver

Diego Oniarti

1 Complessità di una mossa

Metodo naive

$$O\left(\binom{n^2}{m} \cdot n^2\right)$$
$$n^2 = \#celle$$
$$m = \#bombe$$

Permutazioni simili

$$O\left(\binom{n^2}{m}\right)$$
$$n^2 = \#celle$$
$$m = \#bombe$$

2 Algoritmi di risoluzione

L'idea è di implementare diversi algoritmi di risoluzione e di provarli tutti in ordine di complessità crescente. Se uno degli algoritmi riesce a far progredire la partita inizia il processo di nuovo sulla board ottenuta.

2.1 Trivial cells

Alcune celle sono trivialmente libere o bombe. Questo nel caso il numero di una cella sia il numero di flag o celle libere circostanti

Algorithm 1: Trivial cells

```
forall c ∈ celle do
    if c.hidden then
        continue;
    end
    n ← c.num();
    neigh ← c.neighbors().filter(hidden);
    if n = |neigh| then
        forall f ∈ neigh do
            f.flag();
        end
        continue;
    end
    if n = |neigh.filter(flagged)| then
        forall f ∈ neigh.filter(!flagged) do
            f.click();                          // Counts as progress
        end
    end
end
```

2.2 Set theory

L'implementazione di questo algoritmo assume che sia già stato svolto *Algo.1*

2.3 Brute-force

Questo algoritmo è molto pesante, quindi viene usato solo se $\binom{celle\ libere}{bombe}$ si tiene sotto un certo limite

Algorithm 2: Brute-force method

```
p_count ← 0;
forall p ∈ Permutations do
    if !is_possible(p) then
        continue;
    end
    c_count++;
    forall b ∈ p do
        b.count++;
    end
end
progress ← false;
forall c ∈ celle do
    p ← c.count / p_count;
    if p == 1 then
        p.flag() ;                          // Reduce number of bombs
        progress ← true;
    end
    if p == 0 then
        p.click() ;                         // Expand neighbors if possible
        progress ← true;
    end
end
```