Structure de la classe Graph

```
1:  Classe Graph
2:  Attributs :
3:      V : liste des sommets
4:      E : liste des arêtes (paires de sommets)
5:      G_nx : graphe networkx (pour l'affichage)
6:
7:
8:  Fonction GET_NEIGHBORS(v)
9:      l ← ensemble vide
10:     Pour chaque e dans E Faire
11:         Si v = e[0] alors
12:             Ajouter e[1] à l
13:         Si v = e[1] alors
14:             Ajouter e[0] à l
15:     Retourner l
16:
17: Fonction GET_NEIGHBORS_DISTANT(v, distance)
18:     l ← {v}
19:     Si distance ≤ 0 alors
20:         Retourner l
21:     temp ← get_neighbors(v)
22:     l ← l ∪ temp
23:     Pour chaque vertice dans temp Faire
24:         l ← l ∪ get_neighbors_distant(vertice, distance-1)
25:     Retourner l
26:
27: Fonction DIAMETRE
28:     len_max ← 0
29:     Pour chaque i dans V Faire
30:         Pour chaque j dans V Faire
31:             d ← longueur du plus court chemin entre i et j
32:             Si d > len_max alors
33:                 len_max ← d
34:     Retourner len_max
35:
36: Fonction BORNE
37:     Retourner min(ceil(√|V|), diametre())
```

Structure de la classe Etat

```
1:  Classe Etat
2:  Attributs :
3:      B : ensemble des sommets brûlés
4:      NB : ensemble des sommets non brûlés
5:      G : graphe associé
6:      n : numéro de la séquence (étape courante)
7:      B_v : liste des sommets brûlés par chaque balle
8:      C : liste des centres des balles
9:      id : identifiant unique
10:     score : score de l'état
11:
12: Procédure INIT(NB, B, G, n, B_v, C)
13:     self.B ← copie de B
14:     self.NB ← copie de NB
15:     self.G ← G
16:     self.n ← n
17:     self.B_v ← B_v
18:     self.C ← C
19:     self.id ← nouvel identifiant unique
20:     score ← 0
21:     Pour chaque i dans [0, |B_v|) Faire
22:         score ← score + SCORE_SOMMET(self.C[i], i)
23:     self.score ← score − DIAMETRE_NB( )^2
24:
25:
26: Fonction SCORE_SOMMET(sommet, n)
27:     Si n = 0 alors
28:         Retourner 1
29:     Sinon
30:         Retourner |voisins à distance n| / n^2
31:
32: Fonction DIAMETRE_NB
33:     len_max ← 0
34:     Pour chaque i dans NB Faire
35:         Pour chaque j dans NB Faire
36:             d ← longueur du plus court chemin entre i et j dans G
37:             Si d > len_max alors
38:                 len_max ← d
39:     Retourner len_max
```

Structure de la classe Burning_Number

```
1:  Classe Burning_Number
2:  Attributs :
3:      G : le graphe associé
4:      Etat : l'état courant
5:
6:  Procédure INIT(G)
7:      self.G ← G
8:      self.Etat ← nouvel Etat initial (tous sommets non brûlés)
9:
10:
11: Fonction ETAT_INITIAL
12:     Retourner nouvel Etat initial (tous sommets non brûlés)
13:
14: Fonction ACTIONS(etat)
15:     Retourner ensemble des sommets non brûlés dans etat
16:
17: Fonction PROPAGATION(etat)
18:     Créer copies de NB, B, B_v, C depuis etat
19:     Incrémenter n
20:     Pour chaque balle i dans B_v Faire
21:         Pour chaque sommet v brûlé par la balle i Faire
22:             Pour chaque voisin l de v Faire
23:                 Si l n'est pas déjà brûlé par la balle i alors
24:                     Marquer l comme brûlé
25:                     Retirer l de NB
26:                     Ajouter l à B
27:                     Ajouter l à la balle i
28:     Retourner nouvel Etat avec NB, B, B_v, C, n
29:
30: Fonction SUCC(etat, action)
31:     Créer copies de NB, B, B_v, C depuis etat
32:     Incrémenter n
33:     Si action déjà brûlé alors
34:         Erreur
35:     Ajouter une nouvelle balle centrée sur action
36:     Mettre à jour B, NB, B_v, C
37:     Retourner nouvel Etat
38:
39: Fonction GOAL_TEST(etat)
40:     Retourner Vrai si NB est vide, Faux sinon
```

```
1:  Fonction TRAITER
2:      niveau ← borne du graphe divisé par 2
3:      etat_initial ← etat_initial()
4:      noeud_initial ← Noeud(etat_initial)
5:      L_Noeud ← [noeud_initial]
6:      L_a_traiter ← [noeud_initial]
7:      Tant que L_a_traiter non vide Faire
8:          Courant ← premier de L_a_traiter
9:          Générer les enfants de Courant (propagation puis actions)
10:         Trier les enfants par score
11:         Ajouter la moitié supérieure à L_a_traiter et L_Noeud
12:         Si un enfant est but alors
13:             Vider L_a_traiter
14:     Retourner L_Noeud
15:
16: Fonction NOEUDS_GOAL(L_noeud)
17:     L_goal_noeud ← []
18:     Pour chaque n dans L_noeud Faire
19:         Si goal_test(n.etat) alors
20:             Ajouter n à L_goal_noeud
21:     Trier L_goal_noeud par score décroissant
22:     Retourner L_goal_noeud
23:
```

Structure de la classe Noeud

```
1:  Classe Noeud
2:  Attributs :
3:      Etat : l'état associé au noeud
4:      parent : pointeur vers le noeud parent
5:      action : action menant à cet état
6:      cout : coût pour atteindre ce noeud
7:      n : numéro du coup
8:      depth : profondeur dans l'arbre
9:
10: Procédure INIT(Etat, parent, action, cout, n)
11:     self.Etat ← Etat
12:     self.parent ← parent
13:     self.action ← action
14:     self.cout ← cout
15:     self.n ← n
16:     self.depth ← 0
17:     Si parent existe alors
18:         self.depth ← parent.depth + 1
19:
20:
21: Fonction CHILD_NOEUD(problem, action)
22:     Retourner problem.succ(self.Etat, action)
23:
24: Fonction EXPAND(problem, niveau)
25:     L_Noeud ← liste vide
26:     Courant ← self
27:     Etat_Intermediaire ← problem.propagation(Courant.Etat)
28:     L_actions ← problem.actions(Etat_Intermediaire)
29:     Pour chaque i dans L_actions Faire
30:         Etat_enfant ← problem.succ(Etat_Intermediaire, i)
31:         Noeud_enfant ← Noeud(Etat_enfant, Courant, i, Cou-
    rant.cout + 1)
32:         Ajouter Noeud_enfant à L_Noeud
33:     Trier L_Noeud par score croissant
34:     Retourner moitié supérieure de L_Noeud
35:
```

Programme principal

```
1: G1 ← nouveau Graph vide
2: "Instances/graphe.txt" dans G1
3: B1 ← nouvel objet Burning_Number(G1)
4: L1 ← B1.traiter()
5: L2 ← B1.noeuds_goal(L1)
6: taille_balle_min ← taille minimale des listes C dans L2
7:
8: Afficher le nombre total de noeuds retenus (taille de L1)
9: Afficher le(s) noeud(s) de b(G) minimal :
10:
11: Pour chaque l dans L2 Faire
12:     Si taille de l.get_Etat().get_C()−1 = taille_balle_min alors
13:         Afficher l'état correspondant
14: Afficher le graphe G1
```