

RAPPORT

**Projet de
programmation impérative**

Nom : HOU

Prénom: Qing hua

Janvier 2016

Présentation du projet

Le but de ce projet est d'implanter une plateforme de jeu de type morpion 3D. Ce jeu se présenter sous la forme d'un table de taille $n * n$ dans leauel sont placés des piles de jetons marquées par X ou par un O. Initialement aucun jeton n'est présent.

Le jeu se joue à deux joueurs, l'un possède les jetons X, l'autre les jeton O. Le but du jeu est de faire apparaître une ligne de n jetons de sa couleur.

Fonctionnement du jeu

1. Initialement, le table est vide.

On prend en compte maintenant tous les jetons des piles, et on voit le plateau comme une succession de niveau(pas forcément pleins), en partant du plateau, on a 3 possibilités de faire une ligne.

- Sur une même niveau : une ligne, une colonne ou une diagonale.
- Dans une même pile : n jetons identique consécutifs dans une même pile.
- Une ligne «en escalier montant» : sur une ligne, une colonne, ou une diagonale, on a les même jetons en augmentant de 1 le niveau entre chaque pile. (je pense que le diagonale d'un cube est aussi pris en compte)

2. option de la variante

- Variante « « Vue du dessus » »
- Variante 3D

3. option « séisme »

Cette option est compatible avec les deux variantes precedents.

A chaque changement de tour, chacune des piles du plateau a une probabilité de s'effondrer partiellement.

La probabilité qu'une pile s'effondre est fonction de sa hauteur h et de la largeur n de plateau selon la formule : $1 - 2^{-h/(2n)}$.

Si une pile s'effondre, le nombre de jeton à retirer est un nombre aléatoire compris entre 1 et h.

Interface

1. à gauche : le plateau de jeu vu de dessus (on ne voit donc que le sommet des piles), dans lequel une des cases est sélectionnée.
2. à droite : le contenu de la pile sur la case sélectionnée.
3. Une case du table contiendra soit un espace, soit X, soit 0 ; la case sélectionnée sera entourée de +, les autres d'espaces.
4. En dessous du table se trouvent : une ligne de – de la largeur du table, et en dessous une ligne indiquant de quel joueur c'est le tour, et un invite de choix d'action.

Introduction du programme

Mon programme satisfait tous les demandes du professeur.

D'abord, je crée une table, dans laquelle les elements sont les pointeurs qui orientent à les piles. Le nombre des pointeurs est `p_size`, qui est définié par les joueurs.

Au debut du jeu, une table vide affichie, et il y a un position initiale qui doit être entouré par quatre '+'. Et je choisis le coin supérieur droit. Pour afficher la table, la size de table est $3 * p_size$.

Après, les joueurs vont choisir la variante, il y a 2 variantes, « vue du dessus » et « 3D ». Quand ils choisissent la variante, ils vont décider l'option du séisme.

Ensuit, les joueurs doivent effectuer avec i, j, k, l qui representent «en haut » « à gauche » « en bas » « à droite ».L'affichage est rafraîchi après chaque opération. Et pour rafraîchir l'affichage, j'utilise différent moyens pour system Linux et Windows. Les joueurs peuvent aussi effectuer r pour retirer un jeton , p pour mettre un jeton et q pour quitter le jeu.

Pour enreigistrer la position de jeton actuel, je utilise variable globale x et y. Au debut du jeu, ils sont initialisés avec 1, pour chaque mouvement de jeton avec i, j, k, l, x et y sont changés. Quand effectuer avec r, j'utilise le moyen de depile, et avec p j' utilise empile. Avec q, la circulation de fonction bouger arrête.

Après mettre un jeton, on doit vérifier si un des joueurs gagne en fonction de la variante qu'ils ont choisi. Et parceque il y a aussi une possibilité du séisme, ça te dit une partition des jeton vont disparaître. Donc après le séisme, il faut aussi verifier si un des joueurs gagne.

Pour vérifier les conditions de la victoire, j'utilise les moyens mathématique. En première, je collecte tous les jeton de ce joueur, et je fait tous les compositions de `p_size` jetons d'une même couleur(en index). Ensuite, je fait tous les composition de deux dans chaque ces `p_size` jetons. En plus, pour verifier si ces `p_size` jetons s'accordent avec les conditions de la victoire, il y a deux nécessité :colinéaire et sans cesse. Colinéaire signifie qu'il existe que tous les compositions de deux dans `p_size` jetons ont la même veteur unitaire.Sans cesse signifie qu'il existe une composition de `p_size` jetons, dans laquelle il n'y a pas de jeton de adversaire, pour le dire autrement, la distance plus longe dans ces `p_size` jeton est $(p_size - 1)$, ou $2^{(1/2)} * (p_size - 1)$, ou $3^{(1/2)} * (p_size - 1)$.(la diagnale d'une face et la diagonale d'un cube)

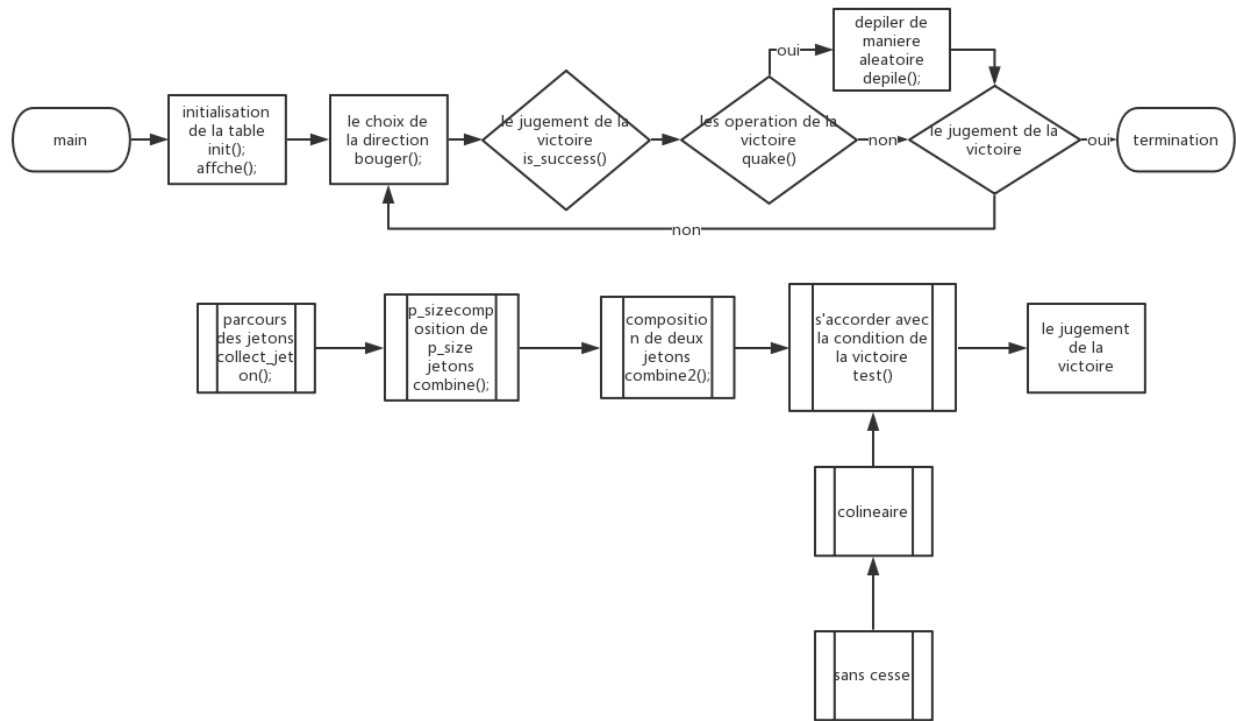
Si ils choisissent séisme, avec la pile devient plus en plus haute, la possibilité qu'il effondre est plus en plus grande. J'utilise le moyen de générer un nombre aléatoire pour décider il s'effondre ou non. Et si il s'effondre, le nombre de jetons qui vont disparaître est aussi dépend d'un nombre aléatoire.

Après le séisme, il doit vérifier si le joueur qui gagne avant toujours gagne après le séisme.

Le graphe de la pensée



Le schéma



Explication de tous les fonctions

1. init_table()

```
void init_table(pile *p1[][p_size]) {
    int i, j;
    for(i = 0; i < p_size; i++)
        for(j = 0; j < p_size; j++)
            p1[i][j] = init_pile();
}
```

Cette fonction est pour initialiser les piles, p[i][j] est pointeur de pile.

2. in_q()

```
int in_q(int i, int j) {
    int k;
    for(k = 0; k < q_size; k++)
        if((i == q[k][0] && j == q[k][1] + 1) || (i == q[k][0] && j == q[k][1] - 1) || (i == q[k][0]
+ 1 && j == q[k][1]) || (i == q[k][0] - 1 && j == q[k][1]) || (i == q[k][0] - 1 && j == q[k][1] +
1) || (i == q[k][0] + 1 && j == q[k][1] + 1) || (i == q[k][0] - 1 && j == q[k][1] - 1) || (i == q[k]
[0] + 1 && j == q[k][1] - 1))
            return 1;
    return 0;
}
```

3. affiche_table()


```

void affiche_table(pile *p[][p_size]) {
    char c;
    int i, j;
    char ch[t_size];
    for(i = 0; i < t_size; i++)
        ch[i] = ' ';
    affiche(p[(x-1)/3][(y-1)/3], ch);
    clear();
    for(i = 0; i < t_size; i++) {
        for(j = 0; j < t_size; j++) {

            if(i % 3 == 1 && j % 3 == 1) {
                c = top(p[(i-1)/3][(j-1)/3]);
                printf("%c", c);
            }
            else if(in_q(i, j))
                printf("*");
            else if((i == x && j == y + 1) || (i == x && j == y - 1) || (i == x + 1 && j ==
y) || (i == x - 1 && j == y))
                printf("+");
            else
                printf(" ");
        }
        printf("\t[%c]\n", ch[i]);
    }
    printf(" ");
    for(i = 0; i < p_size; i++)
        printf("- ");
    char next_user;
    if(step % 2 == 0)
        next_user = 'X';
    else
        next_user = 'O';
    printf("\n%c opt?", next_user);
    printf(" \n");
}

```

Cette fonction affiche la table. Quand il est vide, il y a seulement une position initial qui est entouré par quatre '+'. Quand il y a des piles qui s'effondrent, ils sont entouré par huit '*'. Et il y a quatre '+' entoure la case sélectionnée.

```
C:\Windows\system32\cmd.exe

+
+ +
+
+

[
[
[
[
[
[
[
[
[
[
[
[
[
[
[

- - - - -
X opt?
_
```

4. bouger()

```

void bouger(pile *p[][p_size], char c, int *x, int *y) {
    if(c == 'l' && *y < t_size - 4)
    {
        *y += 3;
        affiche_table(p);
    }
    else if(c == 'i' && *x > 3)
    {
        *x -= 3;
        affiche_table(p);
    }
    else if(c == 'j' && *y > 3)
    {
        *y -= 3;
        affiche_table(p);
    }
    else if(c == 'k' && *x < t_size - 4) {
        *x += 3;
        affiche_table(p);
    }
    else if(c == 'p')
        mettre_jeton(p, x, y);
    else if(c == 'r') {
        depile(p[(x-1)/3][(y-1)/3]);
        step += 1;
        affiche_table(p);
    }
    else
        printf("you have entered a wrong commande!\n");
}

```

Cette fonction enregistre la position d'une case qui bouge. Et si le joueur effectue une commande incorrect, il y a un message d'erreur.

```
C:\Windows\system32\cmd.exe

+
+ +
+

- - - - -
X opt?
_
```

```
C:\Windows\system32\cmd.exe

+
+ +
+

- - - - -
X opt?
]
you have entered a wrong commande!
_
```

```
C:\Windows\system32\cmd.exe

+
+ +
+

X opt?
_
```

```
C:\Windows\system32\cmd.exe

+
+ +
+

X opt?
_
```

5. mettre_jeton()

```

void mettre_jeton(pile *p[][p_size], int *x, int *y) {
    char c;
    if(step % 2 == 0)
        c = 'X';
    else
        c = 'O';
    empile(p[(*x-1)/3][(*y-1)/3], c);
    step += 1;
    affiche_table(p);
    if(var == 'y')
    {
        if (is_success(p, c) && (qua != 'y' && qua != 'Y'))
            exit(1);
        else if(qua == 'y' || qua == 'Y')
        {
            quake(p);
            if(is_success(p, c))
                exit(1);
        }
    }
    else
    {
        if (is_success2(p, c) && (qua != 'y' && qua != 'Y'))
            exit(1);
        if(qua == 'y' || qua == 'Y')
        {
            quake(p);
            if(is_success2(p, 'X'))
                exit(1);
            else if(is_success2(p, 'O'))
                exit(1);
        }
    }
}
}

```

Cette fonction empile et décide ce tour appartient à quel joueur. J'utilise la variable global step. Variable global 'var' enregistre la choix de la variante. Variable global 'qua' enregistre la choix de le séisme. Après le séisme, tous les deux joueurs ont la possibilité de la victoire. Donc il faut verifier 'X' et 'O'. Mais s'il ne choisis pas séisme, il est suffisant de vérifier quel joueur est dans ce tour. (Remarques additionnelle : si et seulement Y/y signifie 'yes', les autre caractère on l'approve tacitement 'non'). Et en mode de non séisme, le jeu termine si un des joueurs gagne. En mode séisme, le jugement de la victoire existe toujours avant et après le séisme.

6. is_success()

```

void is_success(pile *p[][p_size], char c) {
    int con[2000][p_size];
    int b[p_size];
    int xyz_size = collect_jeton(p, c);
    if (xyz_size > p_size - 1) {
        con_size = 0;
        combine(con, xyz_size, p_size, b, p_size);

        int i;
        for (i = 0; i < con_size; i++) {
            int con2[p_size*(p_size-1) / 2][2];
            int con2_size = combine2(con[i], con2, p_size, 0);
            test(con2, con2_size, c);
        }
    }
}

```

Cette fonction vérifie c'est qui gagne. Il y a 3 fonction qu'il appelle, ce sont collect_jeton, combine et combine2.

7. collect_jeton()

```

int collect_jeton(pile *p[][p_size], char c) {
    int xyz_size = 0;
    int i, j, k;
    for(i = 0; i < p_size; i++)
        for(j = 0; j < p_size; j++)
            for (k = p[i][j]->sommet; k > -1; k--)
                if(p[i][j]->a[k] == c) {
                    xyz[xyz_size][0] = i;
                    xyz[xyz_size][1] = j;
                    xyz[xyz_size][2] = k;
                    xyz_size += 1;
                }
    return xyz_size;
}

```

La table xyz[][3] est une variable globale qui garde tous les jetons d'une même couleur, chaque premier index de xyz[][3] représente un jeton. 3 représente les coordonnées tridimensionnelles du jeton. Je parcours tous les piles pour les obtenir. xyz_size est le nombre de jetons qui sont gardés dans xyz[][3].

8. combine()

```

void combine(int con[][p_size], int n, int m, int b[], int M) {
    int i, j;
    for (i = n; i >= m; i--) {
        b[m-1] = i - 1;
        if (m > 1)
            combine(con, i-1, m-1, b, M);
        else {
            for (j = M - 1; j >= 0; j--)
                con[con_size][j] = b[j];
            con_size += 1;
        }
    }
}

```

Cette fonction fait tous les compositions de p_size jetons, et les jetons composés sont gardés dans la table con[][p_size]. Par exemple, si il y a 4 jetons en xyz[][] et p_size égale à 3, la table con[][p_size] garde {(0, 1, 2), (0, 1, 3), (1, 2, 3)}, 0, 1, 2, 3 represente le premier index du chaque jeton dans xyz[][].

9. combine2()

```

int combine2(int con[], int con2[][2], int n, int con2_size) {
    int i, j;
    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++) {
            con2[con2_size][0] = con[i];
            con2[con2_size][1] = con[j];
            con2_size += 1;
        }
    return con2_size;
}

```

Cette fonction fait tous les compositions de deux dans chaque element de con[]. Par exemple, pour con[0], après la composition, j'obtiens {(0, 1), (0, 2), (1, 2)} dans con2[][2], et 0, 1, 2, 3 represente aussi le premier index du xyz[][].

10. test()


```

void test(int con2[][2], int con2_size, char c) {
    double vecteur[con2_size][3];
    int i, j;

    double max_distance = 0.0;

    for (i = 0; i < con2_size; i++) {
        for (j = 0; j < 3; j++)
            vecteur[i][j] = (double)(xyz[con2[i][0]][j] - xyz[con2[i][1]][j]);

        double distance = sqrt(pow(vecteur[i][0], 2.0) + pow(vecteur[i][1], 2.0) +
pow(vecteur[i][2], 2.0));

        if (distance > max_distance)
            max_distance = distance;

        for (j = 0; j < 3; j++)
            vecteur[i][j] = fabs(vecteur[i][j] / distance);
    }
    double moyenne[3];
    for (i = 0; i < 3; i++)
        moyenne[i] = vecteur[0][i];
    int result = 1;
    for (i = 0; i < con2_size; i++) {
        for (j = 0; j < 3; j++) {
            if (fabs(vecteur[i][j] - moyenne[j]) > 0.0001)
                result = 0;
        }
    }

    if (!(fabs(max_distance - (p_size - 1)) < 0.0001 || fabs(max_distance - sqrt(2) * (p_size-
1)) < 0.0001 || fabs(max_distance - sqrt(3) * (p_size-1)) < 0.0001))
        result = 0;

    if (result == 1) {
        printf("\n%c is winner\n", c);
        exit(1);
    }
}

```

Cette fonction vérifie s'il y a une composition dans `con[][p_size]` s'accorder avec les conditions de la victoire. Je conclus que si une composition s'accorder avec ces deux conditions au dessous : colinéaire et sans cesse. Le joueur va gagner.

Pour colinéaire, chaque element dans `con2[][2]` represente un vecteur, si tous les elements dans `con[][2]` ont un même vecteur unitaire, ça te dit que ces `p_size` elements sont colinéaires. Mais c'est pas suffisant parce que s'il a un jeton de adversaire exieste en cette ligne, ils ne s'accordent pas avec les conditions de la victoire. Donc il faut calculer la distance plus longe entre ces `p_size` jeton. Si la distance plus longe est $(p_size - 1)$, ou $2^{(1/2)} * (p_size - 1)$, ou $3^{(1/2)} * (p_size - 1)$. Ça signifie que ils s'accordent avec les conditions de la victoire. Comme dessous :

```
C:\Windows\system32\cmd.exe

X
X 0
X 0
X 0
+
+X+ 0
+
- - - - -
0 opt?
X is winner
请按任意键继续. . .
```

```
C:\Windows\system32\cmd.exe

X X X X +X+
+
0 0 0 0
+
- - - - -
0 opt?
X is winner
请按任意键继续. . .
```

```
C:\Windows\system32\cmd.exe

X
O X
O X
O X
O X+
+
O +X+
+
- - - - -
O opt?
X is winner
请按任意键继续. . .
```

```
C:\Windows\system32\cmd.exe

+
+X+
+
O
- - - - -
O opt?
X is winner
请按任意键继续. . .
```

11. quake()

```

void quake(pile *p[][p_size]) {
    int i, j, h, q;
    double probabilite;
    for (i = 0; i < p_size; i++) {
        for (j = 0; j < p_size; j++) {
            if (! pile_est_vide(p[i][j])) {
                h = p[i][j]->sommet + 1;
                probabilite = 1.0 - pow(2.0, -1.0*h / (2.0*p_size
                    ));
                if (probabilite > 0.1 * (double)p_size) {
                    printf("\nEarthQuake!!! Boom!!!\n");
                    int count = rand() % h + 1; // [1, h]
                    for (q = 0; q < count; q++)
                    {
                        depile(p[i][j]);
                    }
                    affiche2_table(p);
                }
            }
        }
    }
}

```

12. is_success2()

```

void is_success2(pile *p[][p_size], char c) {
    int con[2000][p_size];
    int b[p_size];
    int xyz_size = collect_jeton2(p, c);
    if (xyz_size > p_size - 1) {
        con_size = 0;
        combine(con, xyz_size, p_size, b, p_size);

        int i;
        for (i = 0; i < con_size; i++) {
            int con2[p_size*(p_size-1) / 2][2];
            int con2_size = combine2(con[i], con2, p_size, 0);
            test(con2, con2_size, c);
        }
    }
}

```

Cette fonction est similaire avec is_success, mais c'est pour la variante vue du dessus.

13. collect_jeton2()

```

int collect_jeton2(pile *p[][p_size], char c) {
    int i, j;
    int xyz_size = 0;
    for(i = 0; i < p_size; i++)
        for(j = 0; j < p_size; j++)
        {
            if(top(p[i][j]) == c)
            {
                xyz[xyz_size][0] = i;
                xyz[xyz_size][1] = j;
                xyz[xyz_size][2] = 0;
                xyz_size += 1;
            }
        }
    return xyz_size;
}

```

xyz[][3] ici garde tous les jetons d'une même couleur, mais je parcours seulement la sommet du chaque pile, et je suppose que le haut des jetons gardés sont 0. Dans cette façon, je peut reutiliser les fonctions que j'ai crée.

14. clear()

```

void clear(void) {
#ifdef __linux
    system("clear");
#elif defined _WIN32 || _WIN64
    system("cls");
#endif
}

```

Cette fonction rafraîchit l'affichage, pour différents systems il faut utiliser différentes commandes. Mon ordinateur est Win64, mais il faut vérifier dans Linux, donc j'écris tous les deux.

15. empty_table()

```

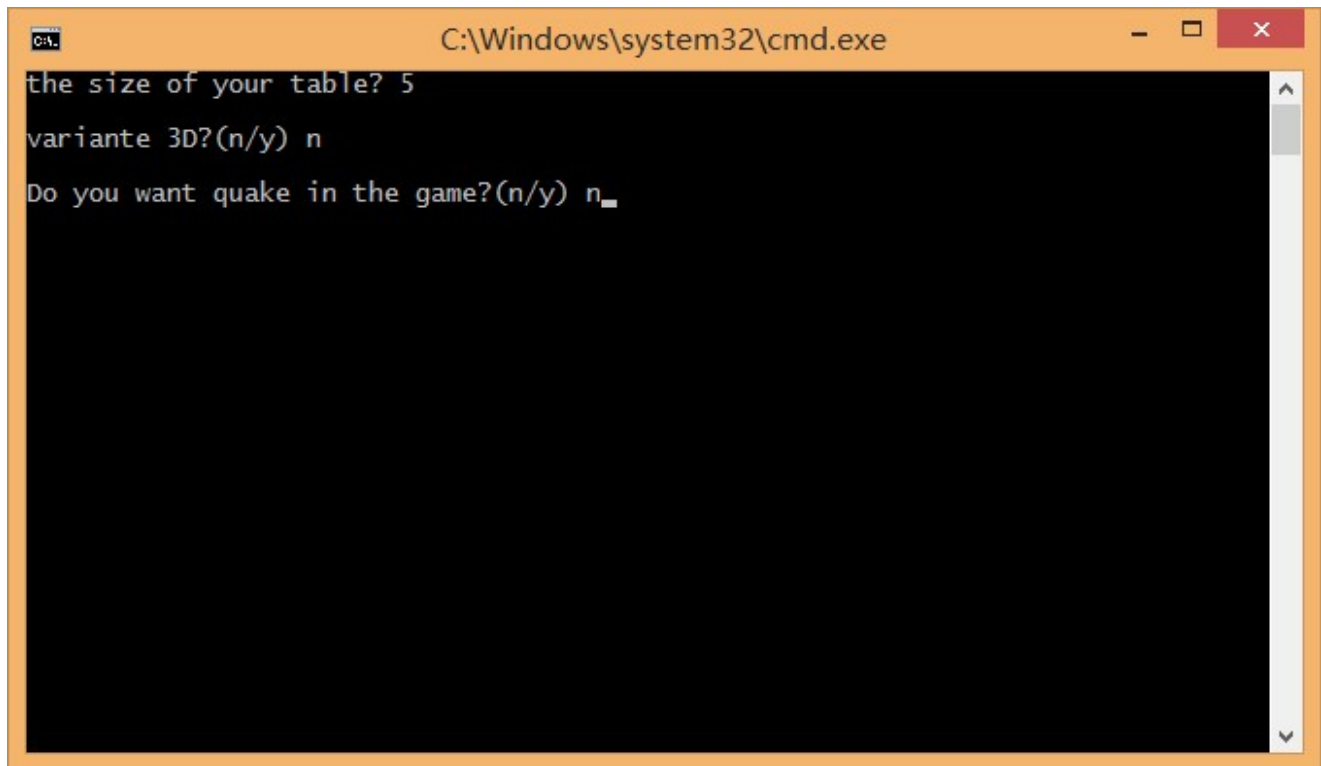
void empty_table(pile *p[][p_size]) {
    int i, j;
    for(i = 0; i < p_size; i++)
        for(j = 0; j < p_size; j++)
            free(p[i][j]);
}

```

Cette fonction free tous les pointeurs parce que j'utilise malloc quand j'initialise les piles.

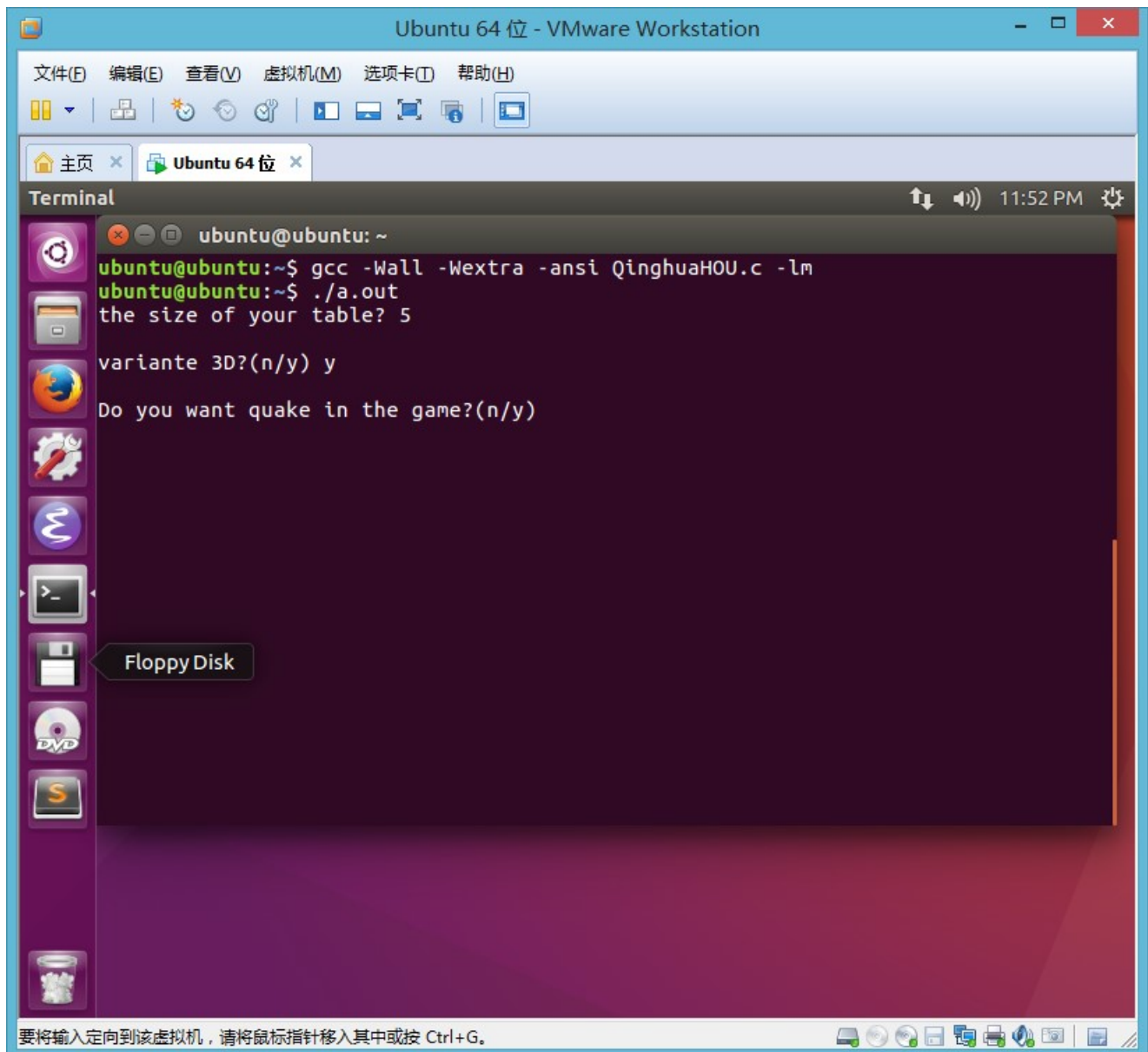
Exécution du programme(*compiler bien en Windows et Linux*)

1. input la longueur de la table, choisir la variante et décider séisme.



A screenshot of a Windows command prompt window. The title bar is orange and displays 'C:\Windows\system32\cmd.exe'. The command prompt has a black background with white text. The text shows the program's execution flow: 'the size of your table? 5' followed by a new line, 'variante 3D?(n/y) n' followed by a new line, and 'Do you want quake in the game?(n/y) n' followed by a cursor. A vertical scrollbar is visible on the right side of the command prompt area.

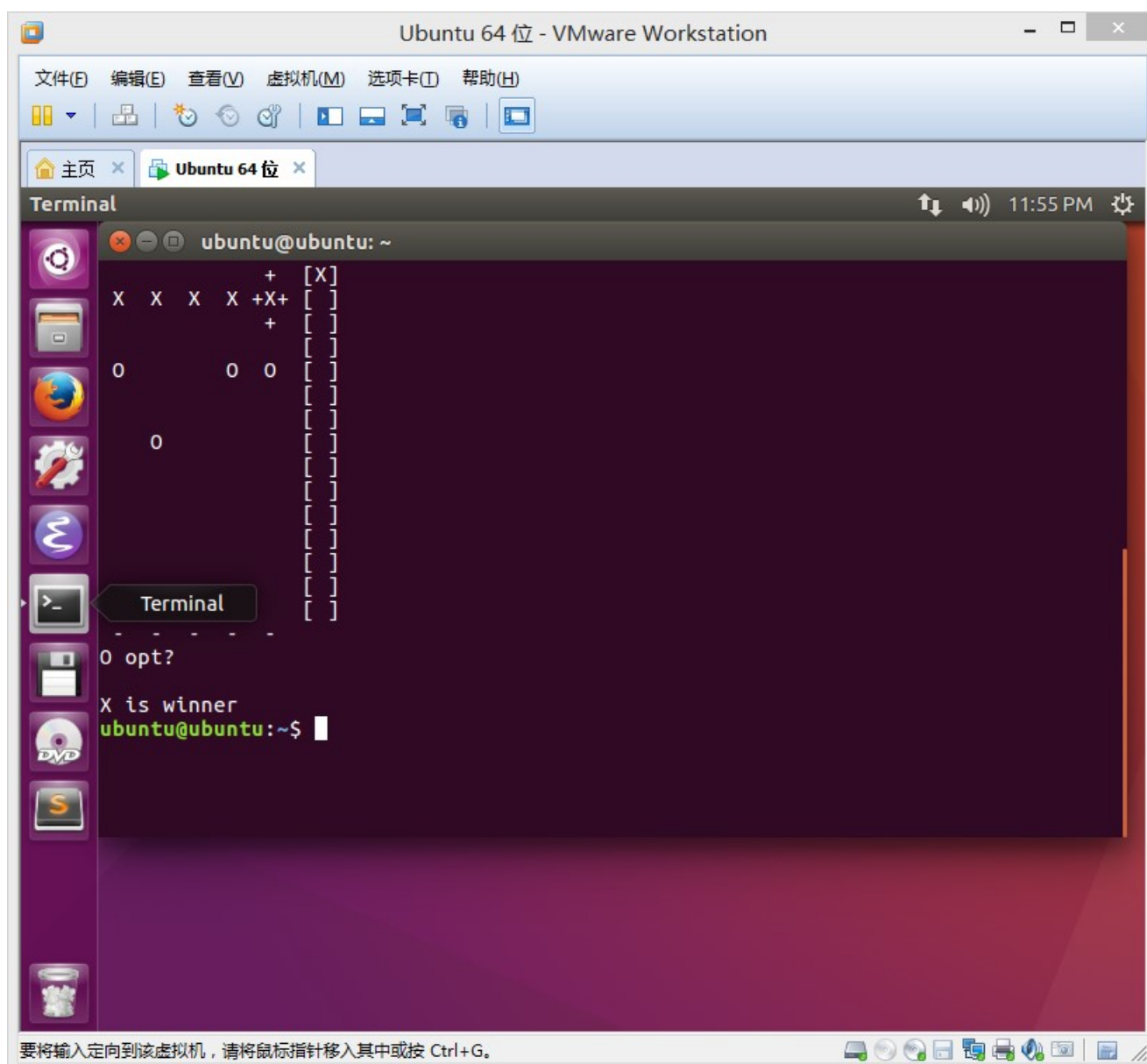
```
C:\Windows\system32\cmd.exe
the size of your table? 5
variante 3D?(n/y) n
Do you want quake in the game?(n/y) n
```



2. plusieurs possibilités de la victoire

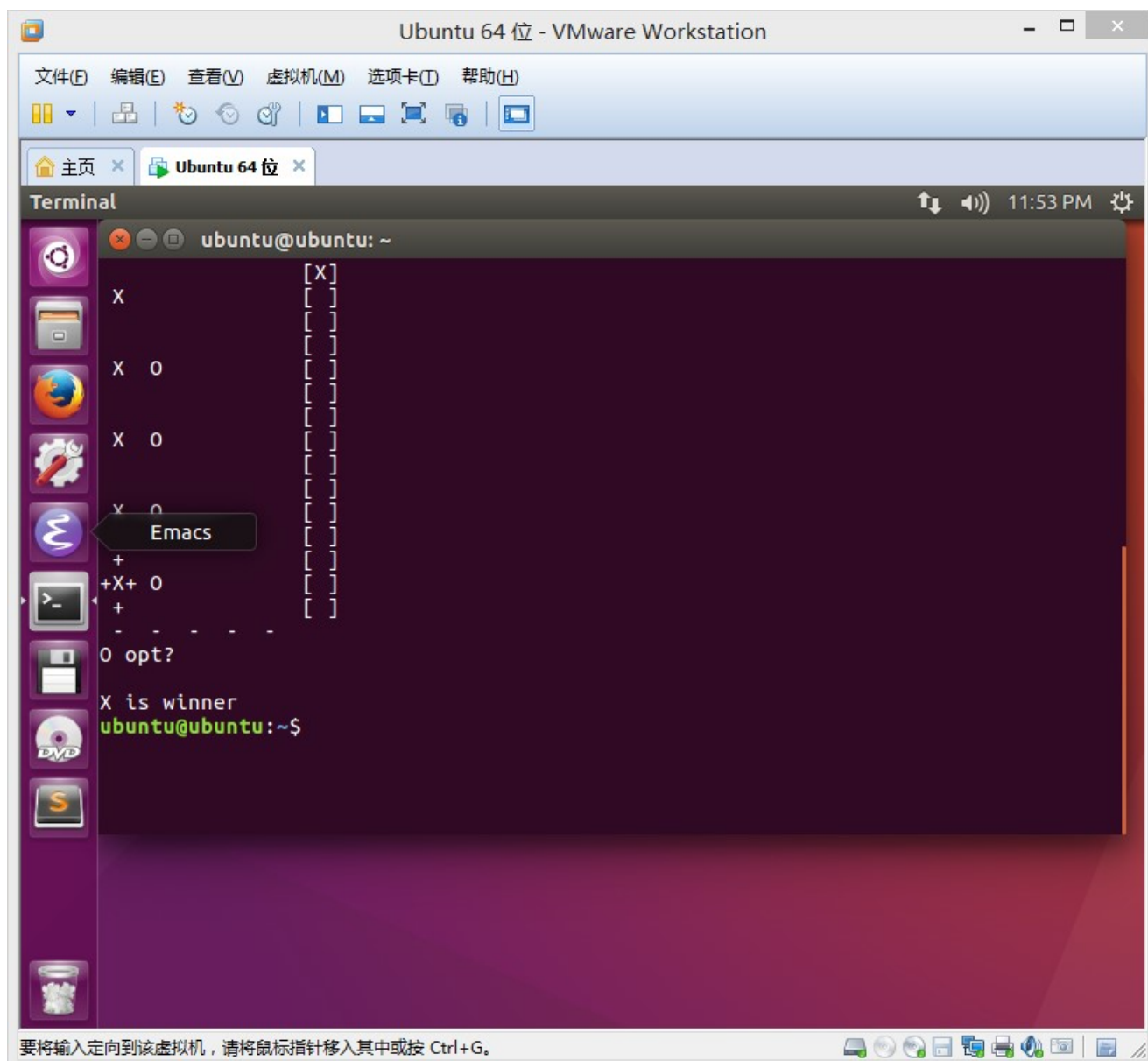
- « vue du dessus » et « non séisme »

[illegible]



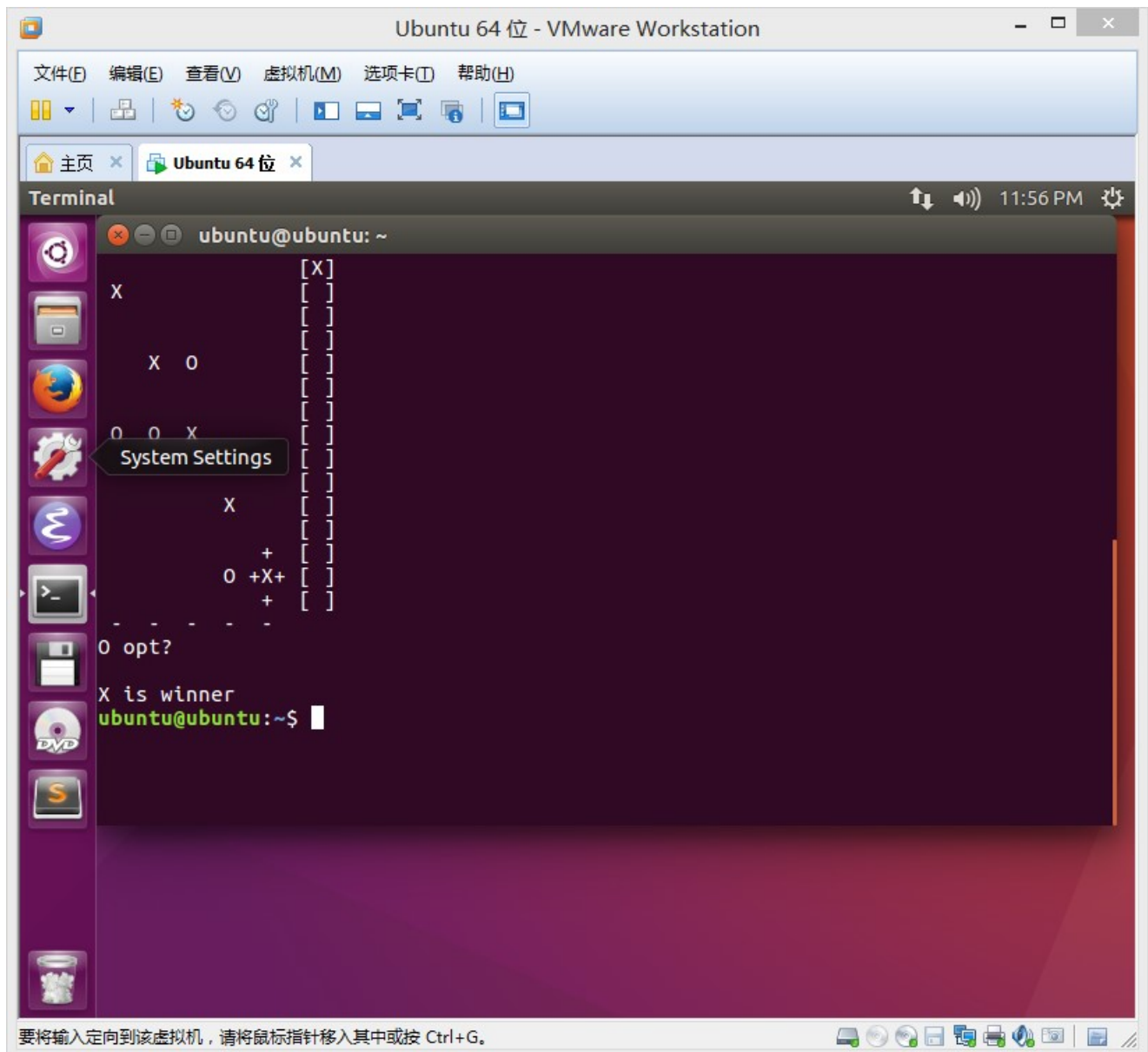
```
C:\Windows\system32\cmd.exe

X
X 0
X 0
X 0 0
+
+X+
+
- - - - -
0 opt?
X is winner
请按任意键继续. . .
```

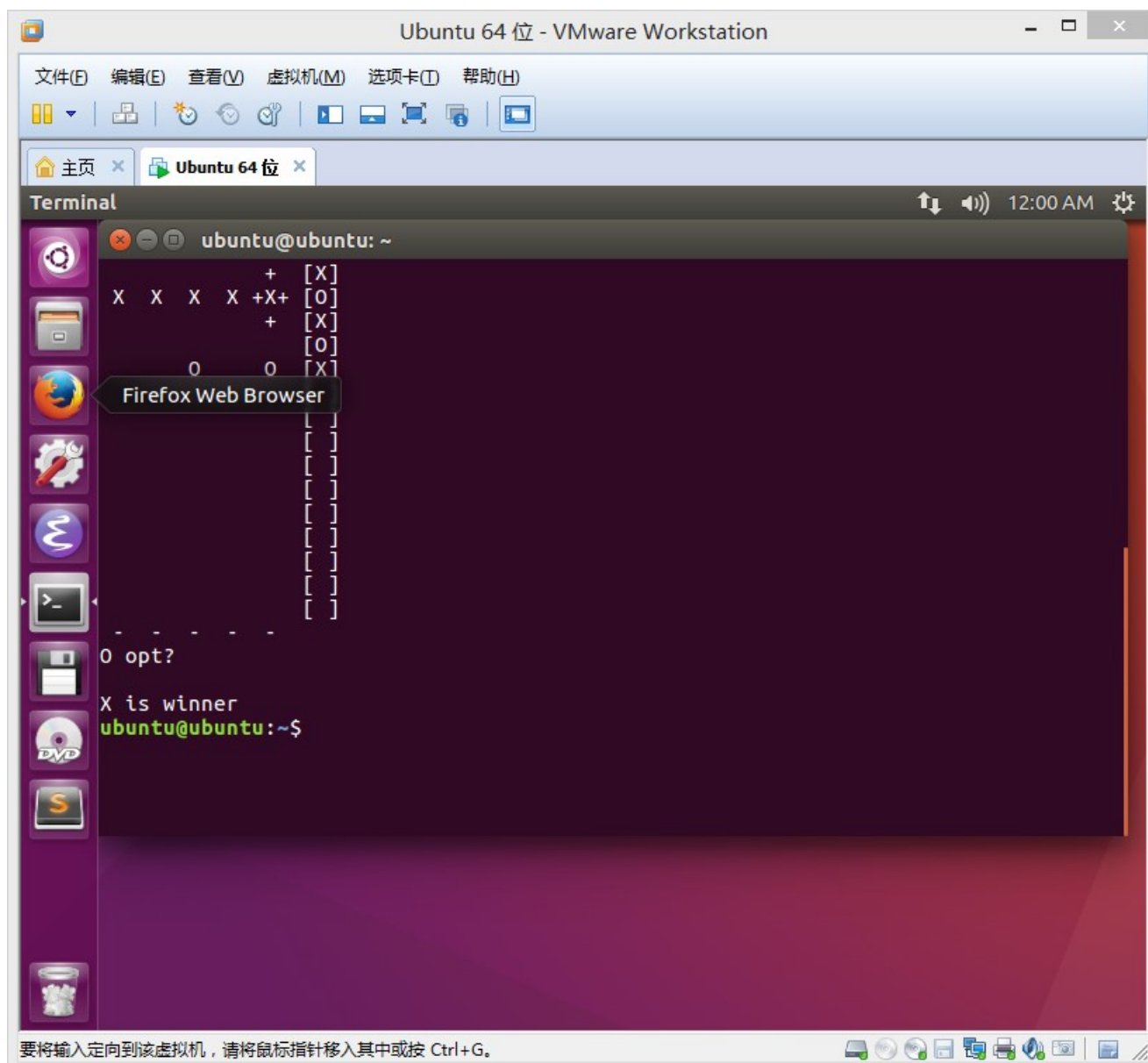


```
C:\Windows\system32\cmd.exe

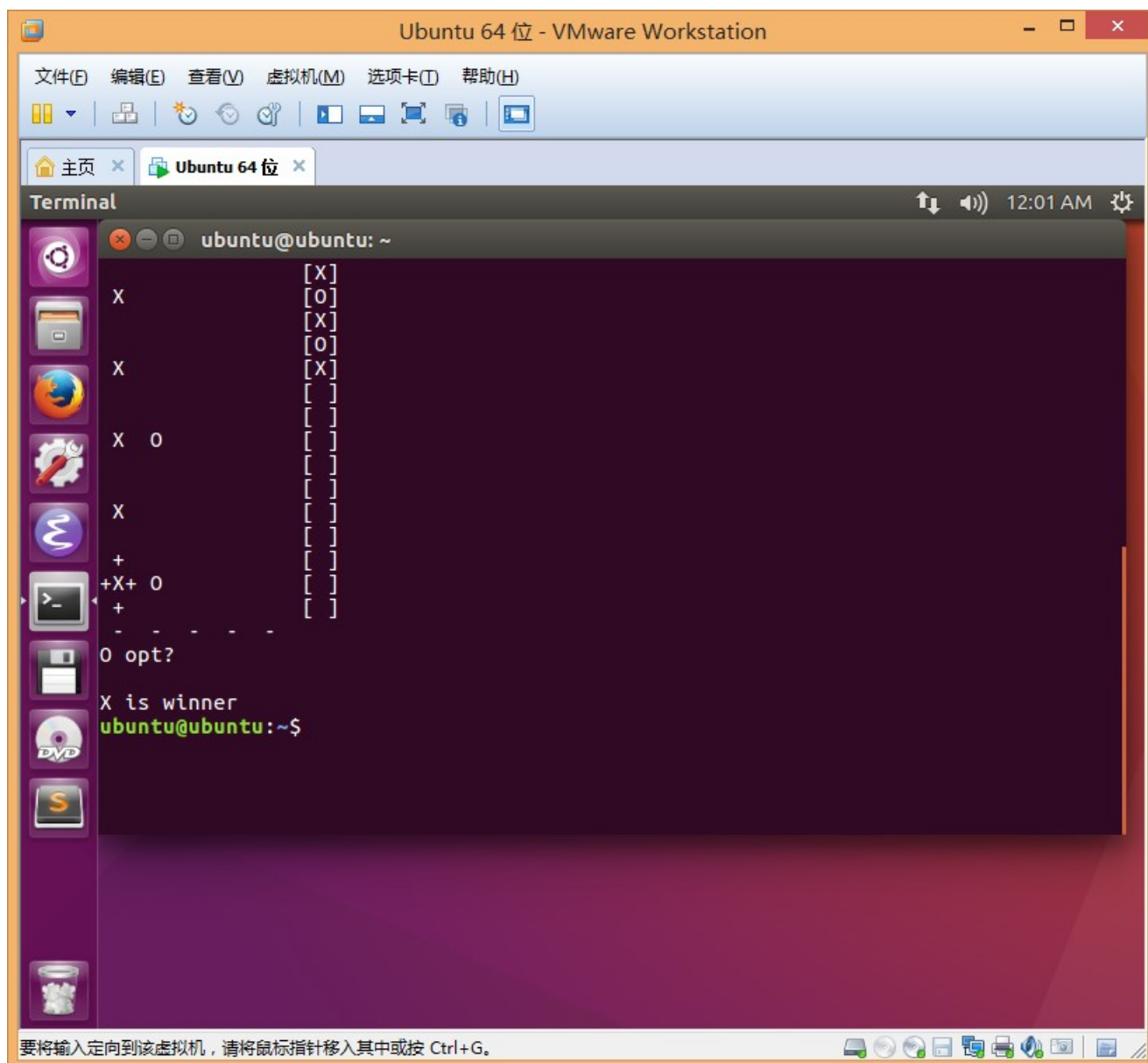
X
O X
O X
O X O
+
+X+
+
- - - - -
O opt?
X is winner
请按任意键继续. . .
```



- « 3D » et « non séisme »



La hauteur de chaque jeton de gauche à droite est 1, 2, 3, 4, 5.

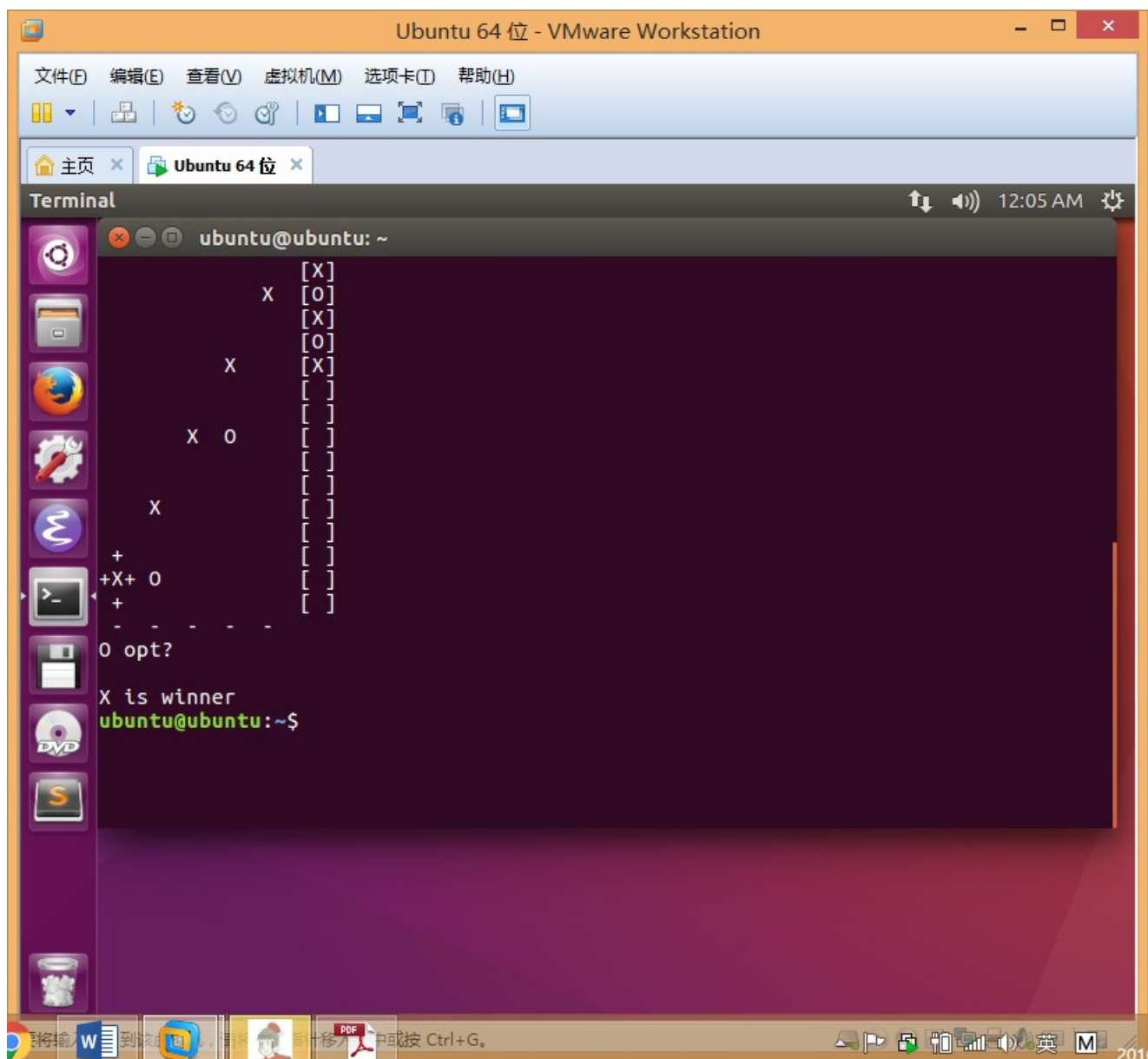


(diagonale d'une face)

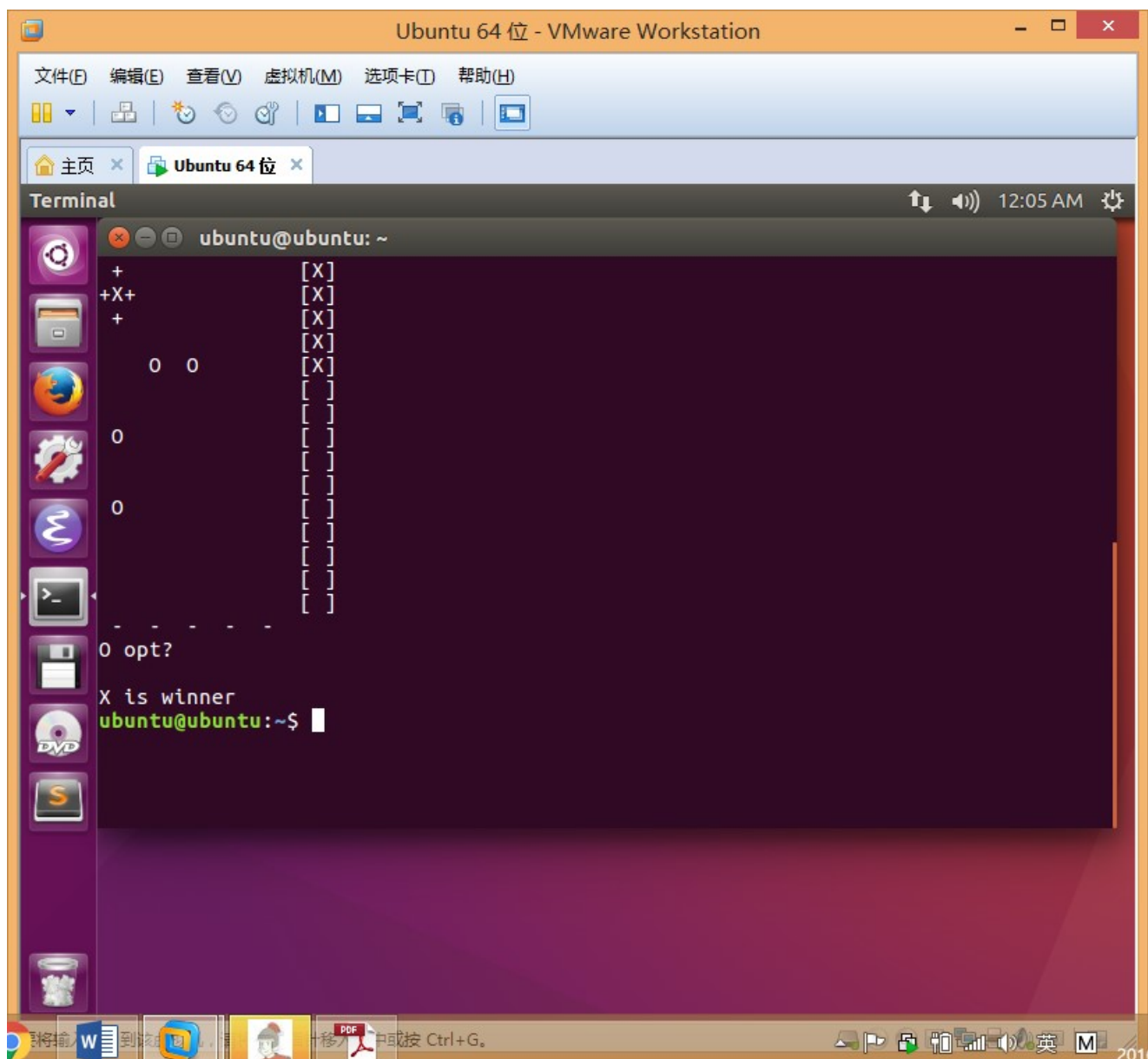
```
C:\Windows\system32\cmd.exe

X
  X
    O X
      X O
        +
        +X+
        +
- - - - -
O opt?
X is winner
请按任意键继续. . .
```

(diagonale d'un cube)



La hauteur de chaque du haut gauche en bas droite est 1, 2, 3, 4, 5.



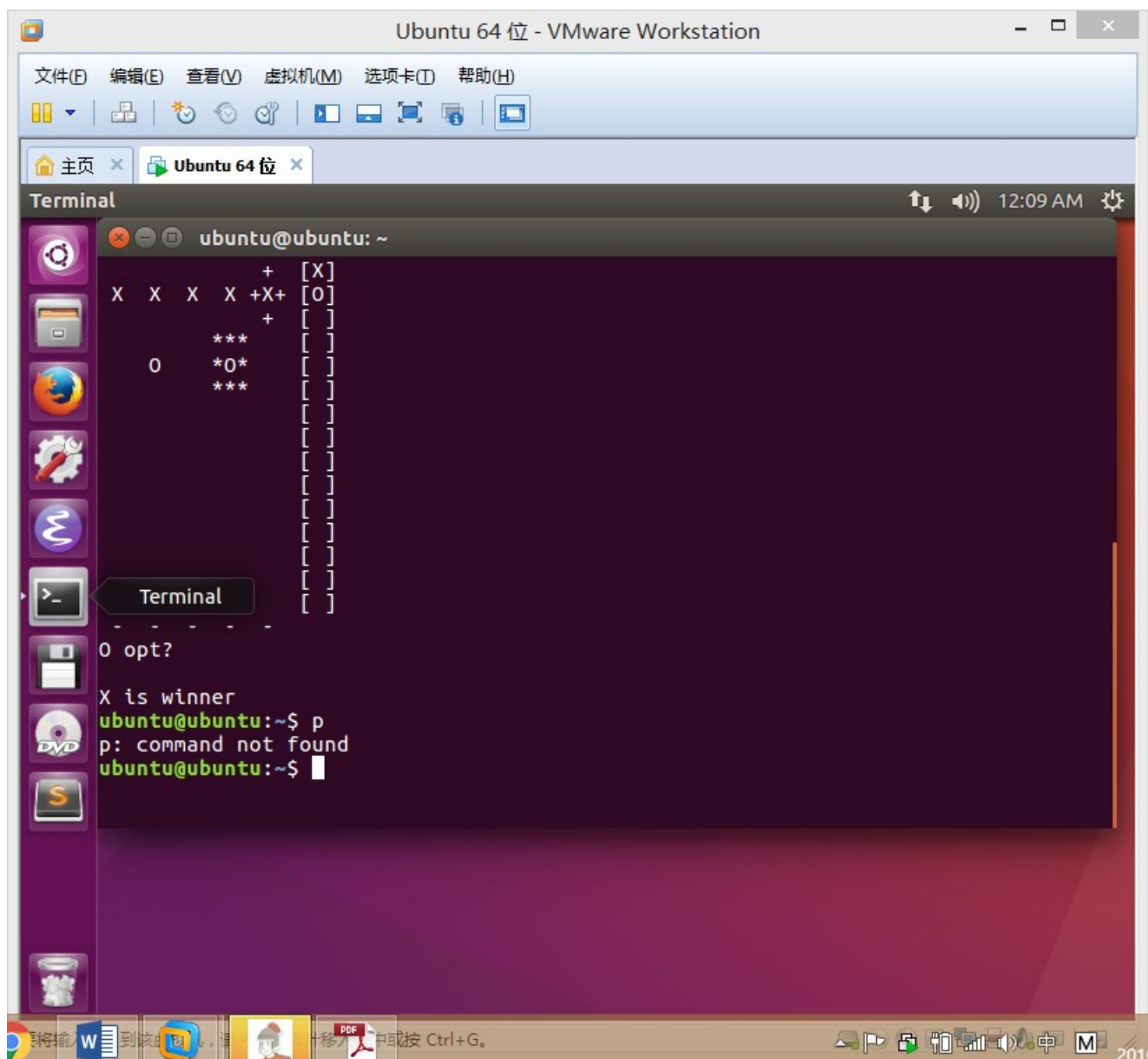
Il y a 5 jeton de même couleur dans une pile.

- « vue du dessus » et « séisme »

```
C:\Windows\system32\cmd.exe

  X
X  X  X  + ***
      +X+*X*
      + ***
      0

- - - - -
0 opt?
X is winner
请按任意键继续. . .
```



Avant le séisme, X ne gagne pas, mais après le séisme, il gagne.

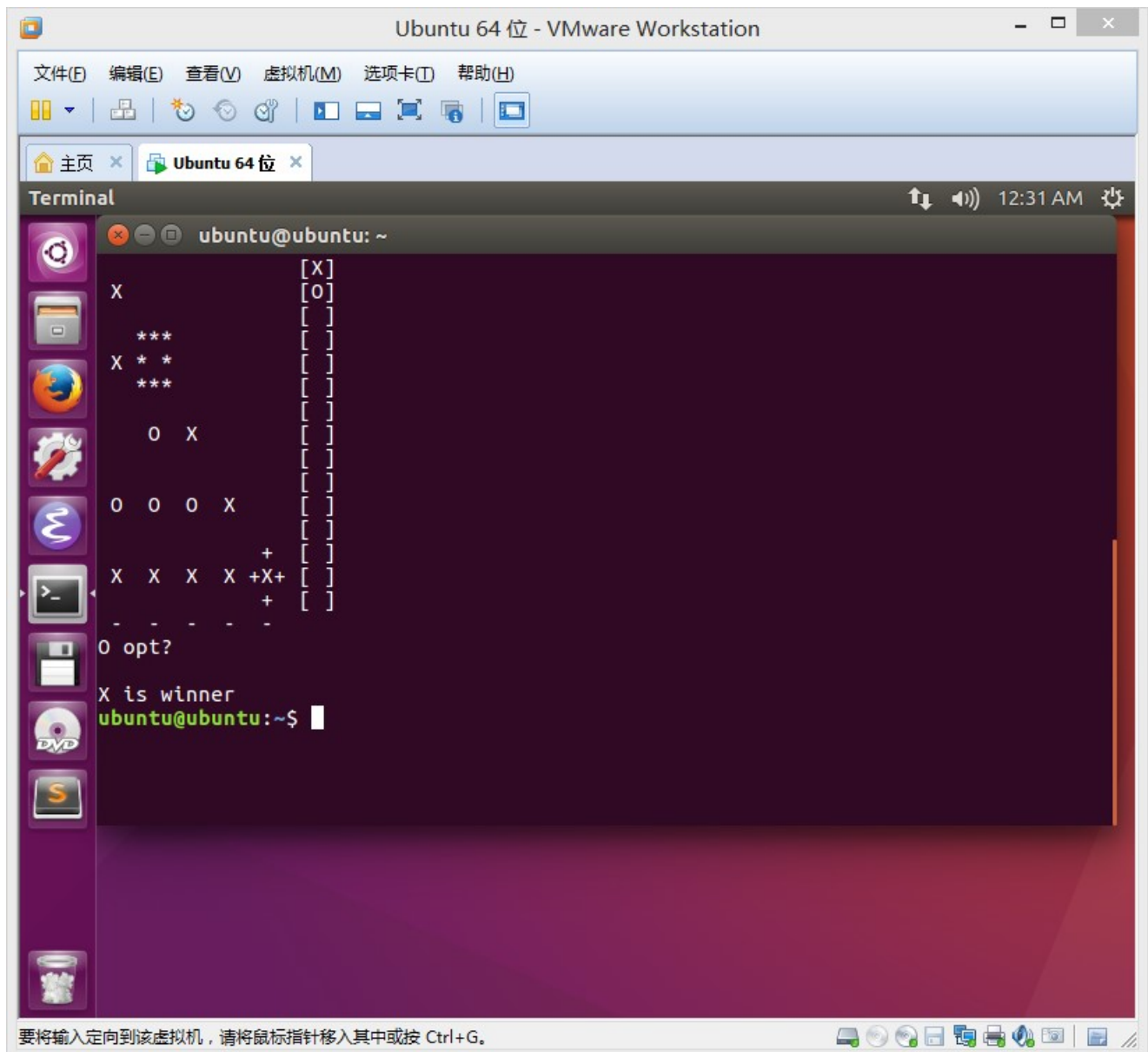
```
C:\Windows\system32\cmd.exe

+
+X+
+
0 X
X X
***
*0* 0 X 0
***
X
- - - - -
0 opt?
X is winner
请按任意键继续. . .
```



```
C:\Windows\system32\cmd.exe

X  O
    X  O
    O  X
        O  X  X
****          +
^O^ X  X  X +X+
****          +
- - - - -
O opt?
X is winner
请按任意键继续. . .
```



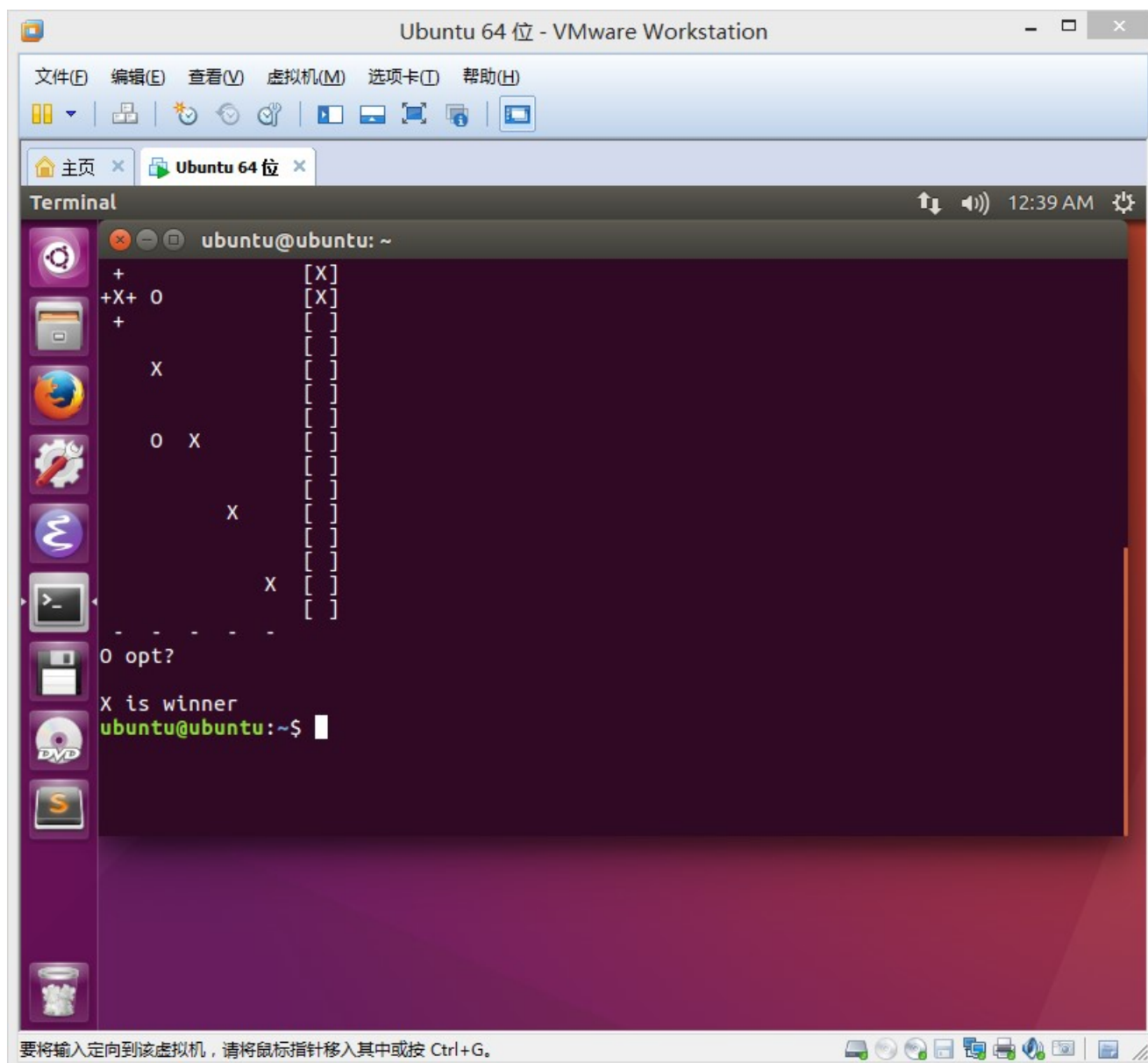
Le 'X' gagne tous les deux simultanément avant et après le séisme, parce il y a deux lignes se forment. (cette situation est un peu difficile à trouver).

- « 3D » et « séisme » (j'ai jamais gagné parce le séisme est fréquente et les conditions de la victoire en 3D est plus exigeante)

```
C:\Windows\system32\cmd.exe

X
  X
    X
      X
        +
        +X+
        +
- - - - -
0 opt?
X is winner
请按任意键继续. . .
```

Dans ce processus, il y a beaucoup des séismes.(cette situation est vraiment difficile à trouver.)



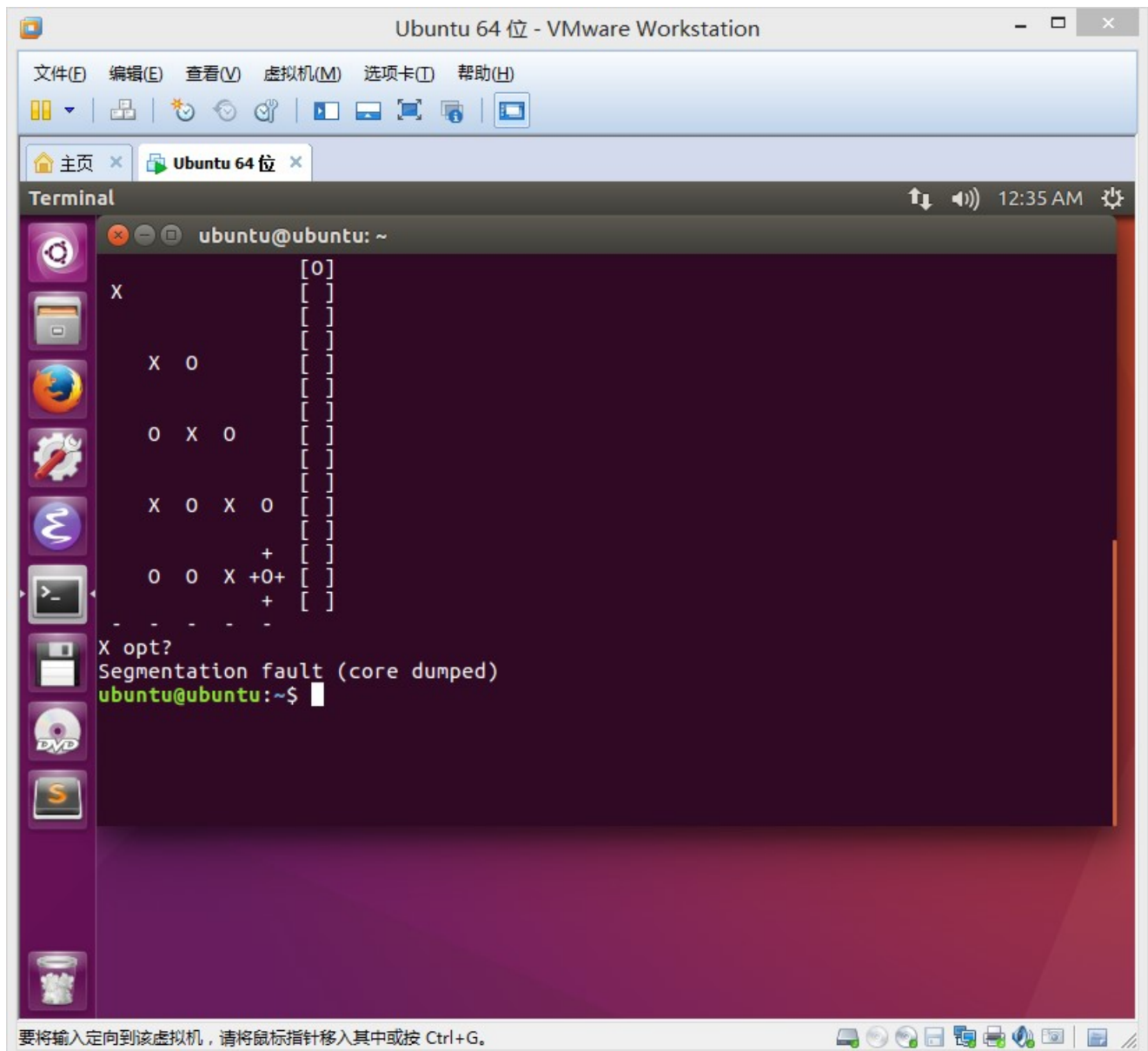
Résumé

1. Il y a beaucoup de experience je conclure sur ce projet:

- La variable globale est pratique, mais il faut faire attention, parceque si la variable globale est dans une circulation, la valeur de la variable globale va changé et ça peut être avoir une influence dans prochaine utilisation de cette variable globale. Donc il faut n'oublier pas la reinitialiser.
- Si on veut verifier si deux valeur de type double sont equivalents, il ne faut pas les comparer directement. Parceque il y a un problème de précision. Donc si la difference entre deux valeur(double) sont assez petite, on les suppose équivalent.
- Si il y a une boucle imbriquée compliquée, l'encapsulation est vraiment utile. C'est une chose difficile à expliquer, mais dans mes codes, la fonction « `int in_q(int i, int j)` » est une exemple.
- `scanf(" %c", &c);` C'est vraiment pratique de mettre une espace avant %c si on voudrait éliminer l'influence de caractère de retour chariot avant.

2. Les limites de mon programme:

- A cause de les limites des tailles des table, c'est possible que il y a une segmentation fault. Donc on doit définir les table assez grandes.



C'est parceque il y a trop de X ou O, le nombre d'une même couleur dépasse 14, donc la nombre des compositions est plus que 2000(le taille de la table `con[2000][]` qu'on a défini)

- Si on choisit la variante de 3D et le séisme, c'est vraiment difficile à gagner, donc je pense le effondrement est un peu fréquent.
- Je pense il y a une façon qui peut améliorer notre projet, dans cette façon, on doit définir les table pas trop grands et peut-être il y a une formule entre la possibilité d'effondrement et la problem du segmentation fault. Ça te dit que si on trouve une formule parfaite de la possibilité qu'une pile s'effondre, on peut aussi réduire la possibilité de segmentation fault. Mais c'est une pensée concernant mathématiques.