new_board

```python
def new_board(n):  1 usage
    board = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n - 1):
        board[i][0] = 1
    for j in range(1, n):
        board[n - 1][j] = 2
    return board
```

Permet de générer le tableau de n lignes avec n colonnes.

Je n'ai pas pu utiliser [[0] * n] * n sinon les colonnes auraient été liées entre elles.

Les deux fors permettent de positionner les pions par défaut.

display_board

```python
def display_board(board, n):  2 usages
    print("Joueur 1 = x", "Joueur 2 = o")
    max_length = 0
    for i in range(n):
        text = str(i + 1) + (len(str(n)) - len(str(i + 1))) * " " + " | "
        for j in range(n):
            text += ("." if board[i][j] == 0 else ("x" if board[i][j] == 1 else "o")) + " " * len(str(n))
        print(text)
        if max_length < len(text):
            max_length = len(text)

    print(" " * (len(str(n)) + 1), end="")
    for i in range(max_length - 3):
        print("-", end="")
    print()

    print(len(str(n)) * " " + 3 * " ", end="")

    for i in range(n):
        print(i + 1, end=" " + " " * (len(str(n)) - len(str(i + 1))))
    print()
```

Même fonction que pour le snort.

select_pawn

```python
def select_pawn(board, n, directions, player):  2 usages
    return int(input("Choisir la ligne d'un pion : ")) - 1, int(input("Choisir la colonne d'un pion : ")) - 1
```

Fonction permettant de récupérer les coordonnées du pion à bouger.

ask_for_directions

```python
def ask_for_direction():  2 usages
    return int(input("Choisir la direction du mouvement (1 pour Nord, 2 pour Est, 3 pour Sud et 4 pour Ouest) : ")) - 1
```

Fonction permettant de récupérer la direction que doit prendre le pion à bouger.

select_move

```python
def select_move(board, n, directions, player, i, j):  2 usages
    m = ask_for_direction()
    while True:
        if len(directions) >= m >= 0:
            if player == 1 and m != 3:  # 4 - 1 (ouest)
                break
            if player == 2 and m != 2:  # 3 - 1 (sud)
                break
        m = ask_for_direction()
    return m
```

Fonction qui demande la direction tant qu'elle n'est pas autorisée ou existante.

Par exemple ici pour le joueur 1 toutes les directions entre 0 et 2 tant que m n'est pas l'ouest.

possible_move

```python
def possible_move(board, n, directions, player, i, j, m):  2 usages
    if board[i][j] != player:
        return False
    x, y = directions[m]
    i += x
    j += y
    if player == 1 and m == 1 and j >= n: # 1 = est
        return True
    if player == 2 and m == 0 and i < 0: # 0 = nord
        return True
    if i >= n or j >= n or j < 0 or i < 0:
        return False
    return board[i][j] == 0
```

Fonction qui permet de déterminer si le mouvement d'un pion depuis une certaine case dans une certaine direction est possible.

Je vérifie d'abord si le pion ciblé est bien celui du joueur.

Ensuite je fais 2 variables temporaires qui vont récupérer les valeurs du tuple de directions. (Par exemple pour le nord -1, 0), il suffit ensuite d'additionner ces valeurs à i et j pour récupérer les nouvelles coordonnées du point qui en résultera.

Je vérifie ensuite pour chaque joueur si le pion peut effectivement sortir dans une certaine direction pour lui.

Sinon si le pion sort ou n'est pas égal à une case vide et donc je retourne False.

move

```python
def move(board, n, directions, player, i, j, m):  1 usage
    x, y = directions[m]
    board[i][j] = 0
    i += x
    j += y
    if i >= n or j >= n or j < 0 or i < 0:
        return
    board[i][j] = player
```

Fonction qui réalise le mouvement du pion.

Même principe au début pour récupérer les nouvelles coordonnées du pion.

On supprime le contenu de la case actuelle puis on applique la direction pour récupérer les nouvelles coordonnées. Si le pion est en dehors, on sort de la fonction pour éviter une erreur.

Enfin on définit la nouvelle case avec le pion du joueur.

get_other_player

```
def get_other_player(player):  1 usage
    return 1 if player == 2 else 2
```

Fonction qui permet de récupérer l'autre joueur (par exemple si le joueur est le 1, la fonction retournera le 2).

win

```
def win(board, n, directions, player):  2 usages
    for i in range(n):
        for j in range(n):
            if board[i][j] == player:
                for m in range(4):
                    if player == 1 and m == 3:
                        continue
                    if player == 2 and m == 2:
                        continue
                    if possible_move(board, n, directions, player, i, j, m):
                        return False
    return True
```

Fonction qui permet de détecter si un joueur a gagné car il n'a plus aucun pion ou s'il ne peut plus bouger.

Pour cela, je vérifie chaque case où le joueur est présent et les 4 directions avec la fonction possible_move.

Bien évidemment je fais en sorte de ne pas tester pour les directions interdites.

dodgem

```
def dodgem(n):  1 usage
    if n <= 1:
        print("Merci de choisir un plateau supérieur à 1")
        return
    directions = ((-1, 0), (0, 1), (1, 0), (0, -1))
    board = new_board(n)
    display_board(board, n)
    current_player = 1

    while not win(board, n, directions, current_player):
        print("Au joueur", current_player, "de jouer")

        i, j = select_pawn(board, n, directions, current_player)
        m = select_move(board, n, directions, current_player, i, j)
        while not possible_move(board, n, directions, current_player, i, j, m):
            i, j = select_pawn(board, n, directions, current_player)
            m = select_move(board, n, directions, current_player, i, j)

        move(board, n, directions, current_player, i, j, m)
        display_board(board, n)

        if win(board, n, directions, current_player):
            break

        current_player = get_other_player(current_player)

    print("Le gagnant est le joueur", current_player)
```

Fonction qui lance le jeu pour une certaine taille de plateau.

A noter que la taille minimale est de 2.

Je définie le tuple des directions, le plateau, j'affiche le plateau et je définie le joueur qui commence.

Je fais ensuite un tant que, avec comme conditions que, tant que le joueur actuel (qui change entre 1 et 2 au fil du temps) n'ait pas gagné, la partie continue.

Dans ce tant que, je récupère les coordonnées du point voulu et la direction tant qu’elles ne sont pas valides et que le mouvement est possible.

Je déplace ensuite le pion et je réaffiche le plateau modifié.

Je vérifie ensuite si le joueur a gagné par la sortie de tous ses pions afin de sortir de la boucle.

J’inverse le joueur actuel pour le nouveau.

Quand on sort de la boucle, j’annonce le bon gagnant.