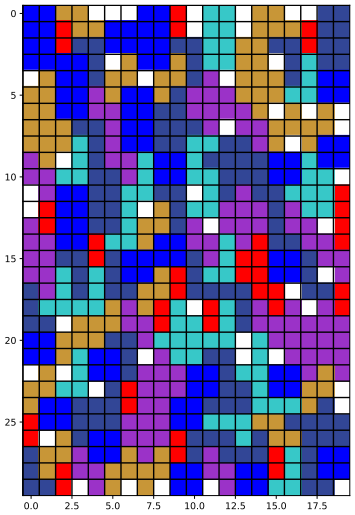
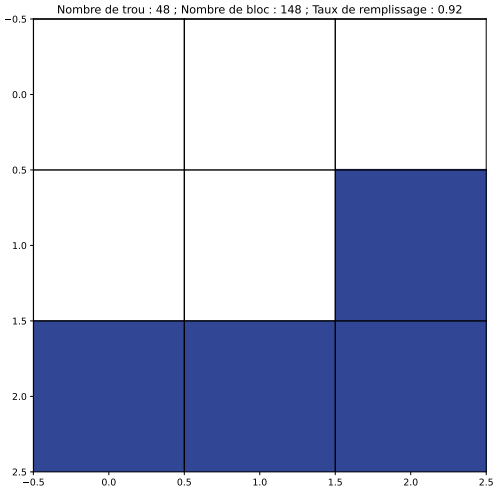





**Description :**

- On dispose d'une grille de taille L×H.
- Des blocs tombent dans un premier temps suivant un ordre défini pour que tout le monde puisse se comparer. Ils seront ensuite choisi aléatoirement.

**Objectif :** Proposer un algorithme de décision qui indique la colonne [en partant de la gauche] (et la rotation) dans laquelle placer le bloc pour un état donné de la grille. L'objectif étant de minimiser le nombre de trou dans la grille.

# Déroulement du jeu

A partir du fichier xxx.py, le jeu se déroule de la façon suivante :

```
# initialise game 20x30
G = game(20, 30, fig=True,list_block = list_block_test)
# Display block list
G.display_all_block_all_config()
# loop until game over
keep_playing = True
while keep_playing:
    s = G.return_game_state()    # get current game state
    a = decision_algo(G, s)      # choose an action
    go = G.update_game_state(a) # update the game
    if go == True:               # check for game over
        print('GAME OVER')
        break

nb_trou, nb_bloc, taux_remplissage = G.print_game_state()
```

A partir d'un état du jeu courant `s` obtenue par la fonction `return_gam_state`, la fonction `decision_algo` doit retourner l'action à effectuer pour positionner le bloc nouveau bloc avant de mettre à jour le jeu avec la fonction `update_game_state`.

### Travail à faire

Il s'agit de définir la fonction `decision_algo` qui retourne l'action à réaliser pour positionner le bloc courant. Pour cela il est possible de faire appel à la méthode `step` de la classe `game` qui simule l'état du jeu après avoir positionné le bloc courant avec l'action indiquée :

```
new_state, game_over = game.step(current_state,action)
```

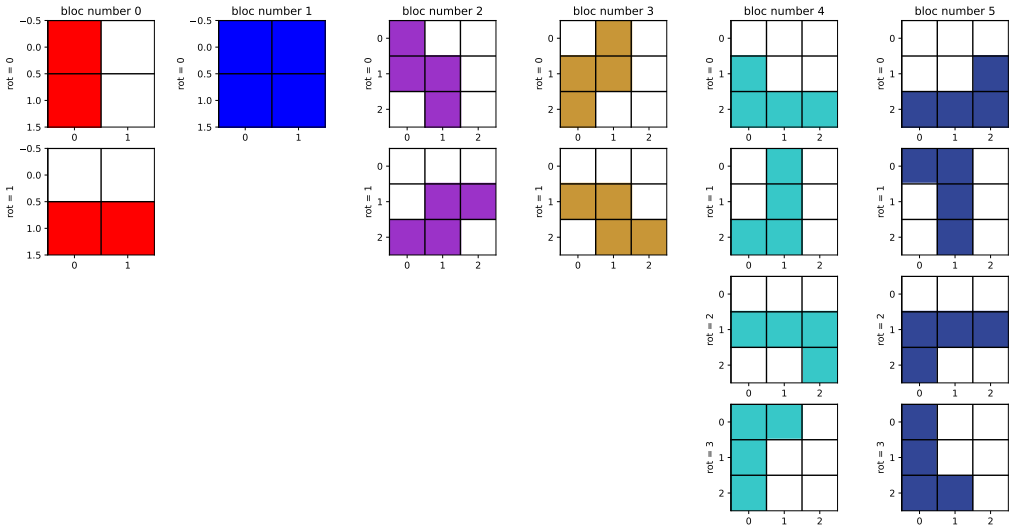
## Quelques explications

Afin de s'aider dans la prise de décision on dispose de l'état du jeu via la classe `state` :

```
class state:
    # game state
    # to be used in the decision algorithm
    # return the game state as seen by the 'player' (not all game info are given)
    def __init__(self, grid, b, list_actions):
        self.grid = grid  # H x L matrix
        self.block = b    # block
        self.list_actions = list_actions # list of available action for the current block
```

Les informations contenues dans cette classe sont :
— La grille est définie par une matrice de taille `H`×`L`.
— Le bloc courant à positionner
— La liste de toutes les actions possibles pour le bloc courant.

Les blocs sont de plusieurs types. Pour afficher tous les blocs possibles vous pouvez utiliser la fonction `G.display_all_block_all_config()`.



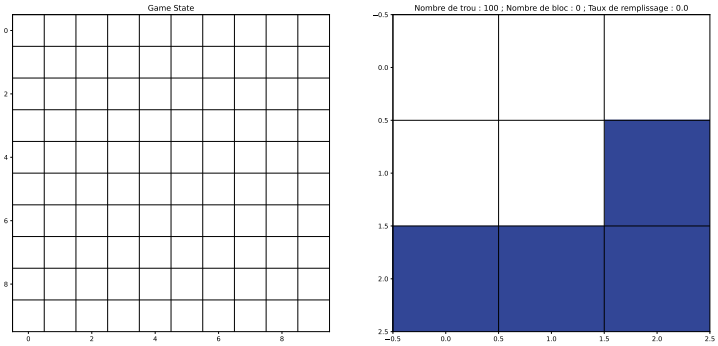

Chaque bloc est défini par son type, sa valeur dans la grille globale, du nombre de rotation de $90^o$ possibles et ainsi des différents `mask` correspondant à sa forme en fonction de la rotation.

```python
class block:
    # define the different types of blocks
    def __init__(self, type):
        self.mask = []
        if type == 0: # 2x1 block
            self.type = 0   # block type in the game process
            self.value = 1  # value in grid for the block
            self.color = [255, 0, 0]
            self.rot = 2    # number of possible rotations
            self.width = [ 1, 2]
            self.height =[ 2, 1]
            self.mask.append(np.array([[1, 0], [1, 0]]))
            self.mask.append(np.array([[0, 0], [1, 1]]))
        elif type == 1: # 2x2 block
```

## Exemple

— Création d'un nouveau jeu :

```python
G = game(10, 10, fig=True)
G.print_game_state()
```



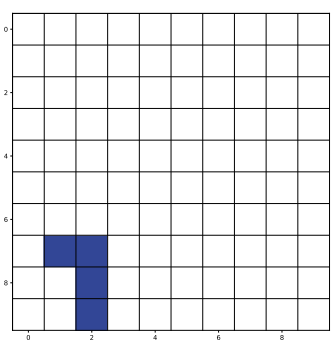

— Positionnement du bloc et mise à jour du jeu, on voit appar :

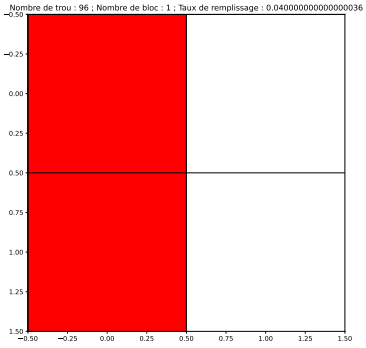
```python
trans = 1 # numéro de la colonne
rot = 1 # rotation de 90 degrés trigo
ag = action_gene(trans, rot) # classe regroupant les 2 actions
G.update_game_state(ag) # mise à jour du jeu
G.print_game_state()
```

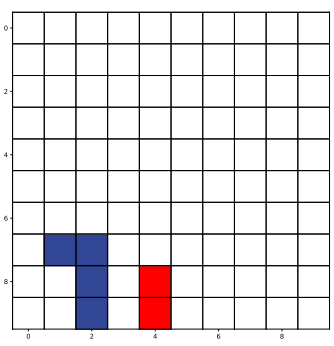
On voit apparaître le nouveau bloc à placer à droite.

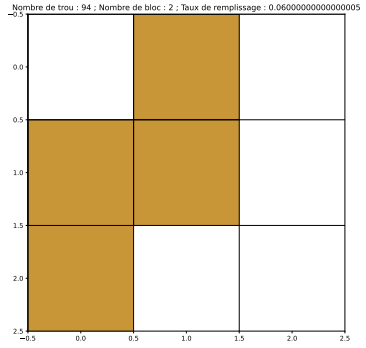
— Positionnement du bloc et mise à jour

```python
    trans = 4 # numéro de la colonne
    rot = 0 # rotation de 90 degrés trigo
    ag = action_gene(trans, rot) # classe regroupant les 2 actions
    G.update_game_state(ag) # mise à jour du jeu
    G.print_game_state()
```




Nombre de trou : 96 ; Nombre de bloc : 1 ; Taux de remplissage : 0.040000000000000036




Nombre de trou : 94 ; Nombre de bloc : 2 ; Taux de remplissage : 0.060000000000000005