TetrisBot Documentation

Version 1.0

F.Muller - L.Ponton

mai 31, 2019

Table des matières

CHAPITRE 1

Le moteur de jeu

1.1 Module tetramino.py

class tetramino.**Tetramino**(*id*, *rotations*, *corners*)

Classe de gestion des pièces

copy()

Renvoie une copie de la pièce

getBottomCells()

Renvoie les coordonnées des cellules les plus en bas

getCorners()

Renvoie les coordonées des coins de la pièce

getLowerCell(*j*)

Renvoie la ligne de la cellule la plus en bas dans la colonne j

rotate(*direction='H'*)

Tourne la pièce dans la direction donnée - "H" : Sens Horaire - "T" : Sens Trigo

setRotation(*i*)

Tourne directement une pièce

toArray()

Renvoie la représentation du bloc sous forme de matrice carrée

1.2 Module board.py

class board.**Board**(*width=10*, *height=22*)

Classe de gestion de la grille

columnHeight(*j*)

Renvoie la hauteur de la colonne j Attention, cette fonction renvoie la hauteur et non l'indice de la dernière pièce

copy()

Renvoie une copie de la grille

decodeFromInt(*n*)
Renvoie un tableau à partir d'un code entier

emptyCell(*i*, *j*)
Vide la cellule (i,j)

encodeToInt()
Renvoie une représentation de la grille sous la forme d'un entier

getBumpiness()
Renvoie la somme des valeurs absolues des différences de hauteurs entre les colonnes consécutives

getCell(*i*, *j*)
Renvoie le contenu de la cellule (i,j)

getColumnHeights()
Renvoie la liste des hauteurs des colonnes

getMaxHeight()
Renvoie la hauteur maximum des pièces du jeu

getNbHoles()
Renvoie le nombre de trous dans la grille (en fait ici juste les cases dominées)

getSumHeights()
Renvoie la somme des hauteurs des colonnes

isCellEmpty(*i*, *j*)
Teste si la cellule (i,j) est vide

isDominated(*i*, *j*)
Teste si une case est vide et est dominée par une case au-dessus

isLineFull(*i*)
Teste si une ligne est pleine

npBinaryRepresentation()
Renvoie un numpy array contenant la grille

printInfos()
Affiche les infos de la grille (pour tests)

processLines()
Enlève les lignes finies et renvoie le nombre de lignes enlevées

removeLine(*i*)
Supprime la ligne i

setCell(*i*, *j*, *value*)
Met value dans la cellule (i,j)

updateStats()
Met à jour tous les paramètres de la grille

1.3 Module tetris_engine.py

class tetris_engine.**TetrisEngine**(*getMove=<function <lambda>>, width=10, height=22, max_blocks=0, base_blocks_bag=[<tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>], temporisation=0, silent=False, random_generator_seed=None, agent_name='', agent_description=''*)
Classe gérant le moteur de jeu

canPlaceBlockDirect(*column*, *rotation*)
Teste si on peut placer directement un bloc dans la colonne et la rotation donnée

copy()
Renvoie une copie de l'environnement

dropBlock()
Fait tomber le bloc en bas

eraseBlock()
Efface le bloc de sa position

generateNewBlock()
Remplace le bloc courant par le suivant et fabrique un nouveau bloc suivant

generateNewBlockBag()
Génère un nouveau sac de pièces

getNewBlock()
Met un nouveau bloc en jeu

getPossibleMovesDirect()
Renvoie la liste de tous les placements directs possibles sous la forme de tuples (column, rotation)

getScoreFromLines(*nb_lines*)
Renvoie le score suivant le nombre de lignes faites

getStrAgentName()
Renvoie la chaîne contenant les nom et la description de l'agent

getStrInfos()
Renvoie une chaîne contenant les informations de la partie

getStrNextBlock()
Renvoie une chaîne pour afficher le bloc suivant

isEndGame()
Teste la fin du jeu

isMoveInGrid(*block*, *new_position*)
Teste si le bloc reste dans la grille

isMoveOnFreeCells(*block*, *new_position*)
Teste si les nouvelles cases occupées par le bloc sont vides

isMoveValid(*block*, *new_position*)
Teste si une position est valide pour un bloc

minimalCopy()
Renvoie une copie minimale de l'environnement (grilles et pièces) pour tester les différents coups

moveBlock(*block*, *new_position*)
Déplace un bloc vers une nouvelle position

moveBlockInDirection(*direction=''*)
Déplace le bloc dans une direction : - "L" : Left - "R" : Right - "" : Vers le bas par défaut

placeBlock(*block*, *position*)
Place un bloc dans une position

placeBlockDirect(*column*, *rotation*)
Place directement un bloc dans la colonne et la rotation donnée

playCommand(*command='N'*)
Joue une commande : - "L" : Move Left - "R" : Move Right - "D" : Drop - "N" : Nothing (descend d'une case) - "H" : Rotate Hours direction - "T" : Rotate Trigo direction - "P :j :r" : Place block in column j with rotation r - "S" : Restart - "Q" : Quit

printRightColumn()
Renvoie la chaîne correspondant au côté droit de l'affichage

rotateBlockInDirection(*direction='H'*)
Tourne le bloc dans une direction : - "H" : Sens Horaire - "T" : Sens Trigo

run()
Boucle principale du jeu

setBlockInitPosition()
Position initiale pour un nouveau bloc

updateTimes(*start_time*)
Met à jour les chronos

tetris_engine.**getrandbits**(*k*) → x. Generates a long int with k random bits.

tetris_engine.**random**() → x in the interval [0, 1).

CHAPITRE 2

Les agents

2.1 Module agent.py

class agent.**Agent**(*name=”*, *description=”*)
La classe de base des agents

allMovesStats()
Renvoie un dictionnaire contenant les stats de chaque mouvement possible Les clefs sont les mouvements et les valeurs sont les stats de ce mouvement

commandFromMove(*move*)
Renvoie la commande d'un mouvement à passer à l'engine

getMoveStats(*move*)
Remplit le dictionnaire contenant les statistiques de la grille après que le mouvement move ait été joué

agent.**benchPlayer**(*player_init*, *nb_samples=100*, *max_blocks=0*)
Réalise un bench de AgentToTest en jouant nb_samples parties

agent.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent.**playGame**(*player_init*, *temporisation=0.1*)
Lance des parties avec l'agent

agent.**plotBenchPlayer**(*player_init*, *nb_samples*, *nb_bars=10*, *max_blocks=0*)
Réalise un bench de AgentToTest en jouant nb_samples parties Affiche les résultats sous la forme d'un histogramme avec nb_bars classes

agent.**random**() → x in the interval [0, 1).

2.2 Module agent_human.py

class agent_human.**AgentHuman**(*temporisation=0*, *silent=False*)
Agent humain

getMove()
Entre un mouvement à jouer

agent_human.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent_human.**random**() → x in the interval [0, 1).

2.3 Module agent_random1.py

class agent_random1.**AgentRandom1**(*temporisation=0.1*, *silent=False*)
Agent aléatoire jouant avec les touches du clavier

getMove()
Renvoie un mouvement de touche aléatoire

agent_random1.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent_random1.**random**() → x in the interval [0, 1).

2.4 Module agent_random2.py

class agent_random2.**AgentRandom2**(*temporisation=0.1*, *silent=False*)
Agent aléatoire jouant directement les pièces

getMove()
Renvoie un mouvement direct aléatoire

agent_random2.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent_random2.**random**() → x in the interval [0, 1).

2.5 Module agent_filtering.py

class agent_filtering.**AgentFiltering**(*temporisation=0.1, silent=False, order=['holes', 'sum_heights', 'bumpiness', 'lines']*)
Agent procédant par filtrage des coups

filterMoves(*stat*, *value*)
Filtre les mouvements en récupérant uniquement ceux dont la stat vaut value

getMove()
Optimisation en filtrant successivement les mouvements suivant les différentes stats

maxStat(*stat*)
Renvoie la valeur maxi d'une stat

minStat(*stat*)
Renvoie la valeur mini d'une stat

agent_filtering.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent_filtering.**random**() → x in the interval [0, 1).

2.6 Module agent_evaluation.py

class agent_evaluation.**AgentEvaluation**(*eval_coeffs=[0.548, 0.5218, 0.6267, 0.1862], temporisation=0.1, silent=False*)
Agent procédant par évaluation des coups

getMove()
Optimisation à partir de la fonction d'évaluation

moveEvaluation(*move*)
Évalue le mouvement move=(j, r)

agent_evaluation.**getrandbits**(*k*) → x. Generates a long int with k random bits.

agent_evaluation.**playGameWithAgentEvaluation**(*coeffs*, *temporisation=0*)
Joue des parties avec l'agent par évaluation et les coeffs donnés

agent_evaluation.**random**() → x in the interval [0, 1).

CHAPITRE 3

Algorithmes génétiques

3.1 Module ag_optimizer.py

class ag_optimizer.**AGOptimizer**(*population_size=20*, *nb_generations=2*, *nb_bits=16*, *max_nb_blocks=5*, *nb_games_played=1*, *proba_mutation=0.05*, *mutation_rate=0.2*, *percentage_for_tournament=0.1*, *percentage_new_offspring=0.3*, *elitism_percentage=0.2*, *vector_encoding='float'*, *parents_selection_method='tournament'*, *old_generation_policy='best'*, *evaluation_criteria='lines'*)

Optimisation des coefficients par algorithme génétique

crossover(*parent1*, *parent2*)

Renvoie le ou les enfants de parent1 et parent2

deleteWorst()

Enlève les moins bons éléments de la population

fitness(*vector*)

Fitness de l'individu : score total sur nb_games_played parties

generateNewOffspring()

Renvoie la nouvelle génération

initPopulation()

Initialise la population

keepOnlyElite()

Garde les meilleurs éléments de la génération précédente

makeNewGeneration()

Crée une nouvelle génération

mutate(*individu*)

Mute un individu

mutateBinVector(*bin_vector*)

Mute un vecteur

mutateFloatVector(*vector*)

Mute un individu (son vecteur)

plotStats()
Courbes de statistiques

process()
Boucle principale de l'optimisation

randomBinaryList()
Renvoie une liste de self.nb_bits chiffres binaires aléatoires

randomBinaryVector()
Renvoie un vecteur binaire aléatoire

randomFloatVector()
Renvoie un vecteur aléatoire normé

scoreOnOneGame(*vector*)
Score sur une partie

sortPopulationDescending()
Trie la population par ordre décroissant de scores

stringOfParameters()
Renvoie la chaine des paramètres de l'algorithme génétique

tournamentSelection()
Sélection par tournoi

updateBinaryIndivdu(*individu*)
Met à jour les paramètres d'un individu à partir de son vecteur binaire

updateScore(*individu*)
Met à jour le score de l'individu

updateStats()
Met à jours les statistiques

wheelSelection()
Sélection d'un individu avec une roulette

ag_optimizer.**binToFloat**(*bits*)
Renvoie la représentation entre 0 et 1 d'une liste binaire Le chiffre des unités étant considéré comme le 1er élément de la liste (ça n'a aucune importance vu qu'on va partir de listes aléatoires)

ag_optimizer.**binVectorToFloat**(*bin_vector*)
Convertit un vecteur de listes binaires en vecteur de float

ag_optimizer.**getrandbits**(*k*) → x. Generates a long int with k random bits.

ag_optimizer.**linearCombination**(*a1*, *vector1*, *a2*, *vector2*)
Renvoie la combinaison linéaire de deux vecteurs

ag_optimizer.**normalize**(*vector*)
Normalise un vecteur

ag_optimizer.**random**() → x in the interval [0, 1).

CHAPITRE 4

Reinforcement learning

4.1 Module tetris_RLenv.py

class tetris_RLenv.**TetrisEnv**(*width=10, height=22, max_blocks=0, base_blocks_bag=[<tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>, <tetramino.Tetramino instance>], random_generator_seed=None, agent_name='', agent_description=''*)

Environnement à la OpenAI Gym pour implémenter le reinforcement learning

getState()

Renvoie l'état de la grille

getStateCode()

Renoie de code entier de l'état de la grille

render()

Affiche le jeu

reset()

Réinitialise l'environnement

sampleAction()

Renvoie une action aléatoire

step(*action*)

Effectue une action (joue un coup) Met à jour les informations (done, state)

tetris_RLenv.**getrandbits**(*k*) → x. Generates a long int with k random bits.

tetris_RLenv.**random**() → x in the interval [0, 1).

4.2 Module qRL_optimizer.py

class qRL_optimizer.**QRLOptimizer**(*width=5, height=5, base_blocks_bag=[<tetramino.Tetramino instance>], max_episodes=2000, max_blocks=500, alpha=0.1, gamma=0.9, epsilon_min=0.01, epsilon_max=1, epsilon_delta=0.001*)

Optimisation par Q-learning sur une configuration simple

getQIndexes()

Renvoie un tableau dans lequel chaque cellule est le nombre de fois qu'elle apparaît dans les indices de la Q-Table avec des Q-Values toutes non nulles

initQValue(*s*)

Initialise la Q-value de l'état s avec des 0 si s n'est pas encore dans la table

learn()

Lance l'apprentissage

play()

Joue la partie avec la Q-table crée

printQIndexes()

Affiche le nombre de fois que chaque cellule apparaît dans les indices de la Q-Table en nuances de gris

reinit()

Réinitialise l'environnement

update(*s*, *a*)

Met à jour la Q-table de l'état s sur une action a

qRL_optimizer.**argmax**(*liste*)

Renvoie l'indice de la valeur max de liste ou un indice aléatoire si la liste ne contient que des 0

qRL_optimizer.**getrandbits**(*k*) → x. Generates a long int with k random bits.

qRL_optimizer.**random**() → x in the interval [0, 1).

qRL_optimizer.**total_size**(*o*)

Renvoie la taille totale d'un objet en mémoire Code récupéré sur : https://code.activestate.com/recipes/577504/ et adapté au projet

CHAPITRE 5

Divers

5.1 Module stats.py

class stats.**Stats**(*data=None*, *filename=''*, *mean_time=0*, *nb_bars=10*, *title=''*)

Représentation statistique des parties

getAllStats()

Récupère tous les indicateurs statistiques

getEffectif()

Renvoie le nombre de données

getMaxi()

Renvoie le maximum des données

getMean()

Renvoie la moyenne des données

getMini()

Renvoie le minimum des données

getQuartiles()

Renvoie un tuple (Q1, Médiane, Q3)

getSigma()

Renvoie l'écart-type des données

histogram(*save=True*)

Crée et affiche l'histogramme

loadData()

Charge les données à partir d'un fichier texte

saveData()

Sauvegarde les données dans un fichier texte

5.2 Module textutil.py

textutil.**boxed**(*text*, *prefix=''*, *window_width=0*, *window_height=0*)

Affiche chaque ligne de texte précédée d'un préfixe dans une boîte de largeur window_width

textutil.**center**(*string*, *length*)
Centre la chaîne string sur la longueur

textutil.**cleanLine**(*string*)
Renvoie la chaine sans les caractères spéciaux de couleur

textutil.**dateNow**()
Renvoie une chaîne avec la date et l'heure courante

textutil.**mergeChains**(*string1*, *string2*)
Fusionne deux chaînes cote à cote pour l'affichage

textutil.**textColor**(*string*, *bg=15*, *fg=232*)
Renvoie une chaîne contenant le texte coloré avec bg pour la couleur de fond et fg pour la couleur du texte. Utilise les codes ANSI/VT100.

Index des modules Python

Index

H

I

K

L

M

N

P

Q

R

S

T

U

W