ÉCOLE POLYTECHNIQUE - ESPCI CONCOURS D'ADMISSION 2019
ENS : LYON, PARIS-SACLAY

Composition d'informatique 2h, filière PC (XELC)

Bilan général des candidats français :

0<=N<4	50	10,8%
4<=N<8	217	46,87%
8<=N<12	159	34,34%
12<=N<16	35	7,56%
16<=N<=20	2	0,43%
Total :	463	100%
Nombre de candidats :	463	
Note moyenne :	7,49	
Ecart-type :	3,08	

Commentaires

Cette année, le sujet portait sur la réalisation d'un jeu de type Tetris avec des cases colorées. La partie I correspondait au lancement du jeu. La partie II portait sur les mouvements des barreaux. Dans la partie III, se trouvait les fonctions de calcul de score, d'élimination des alignements et de tassement. La partie IV considérait une variante du jeu. Finalement, la dernière partie considérait la gestion de la base de donnée des parties et des joueurs. L'énoncé commençait par un ensemble de consignes clairement détaillées, en particulier sur les opérations python autorisées.

Maîtrise des opérations de base

Plusieurs copies montrent une incompréhension inquiétante de l'utilisation des boucles, et en particulier des boucles while. Par exemple, pour écrire while A and B, certaines copies proposent while A if B, ce qui ne fonctionne pas du tout de la même manière. Dans d'autres copies, des boucles while et for imbriquées sont utilisées pour tenter de reproduire le fonctionnement d'un while. Certains candidats rencontrent également des difficultés avec les boucles for. Dans plusieurs copies, des candidats ont voulu modifier la valeur de i à l'intérieur d'une boucle for i in range.... C'est une pratique fortement déconseillée, dangereuse dans la pratique, en dehors de toute règle de bonnes pratiques, et ce quelque soit le langage utilisé. Cela a surtout été le signe d'une incompréhension totale du fonctionnement des boucles for dans chacune des copies concernées.

Il y a beaucoup de confusion sur les différentes opérations python et leurs usages. Certains confondent les opérations range et len, qui n'ont pourtant que peu de rapport. Leurs fonctionnements respectifs étaient pourtant explicités au début du document. Les correcteurs ont également trouvé

beaucoup de confusion entre les opérations break et return. L'opération return termine la fonction ou la procédure (avec ou sans argument) indépendamment du reste du programme. L'opération break permet de sortir de la boucle while ou for contenant l'appel à cette opération. En cas de boucles imbriquées, une seule boucle est impliquée. Cette opération ne peut être appelée que dans une de ces boucles.

Dans les deux premières parties (questions de 1 à 7), plusieurs copies recopiaient systématiquement la grille donnée en argument. C'est une opération coûteuse et inutile qui démontre une incompréhension majeure du fonctionnement d'un programme et de sa complexité. Toujours dans les 7 premières questions, les correcteurs ont rencontré au sein de certaines fonctions ou procédures des variables 'jumelles', par exemple une variable i initialisée à 0, une variable j initialisée à n, ces deux variables étant respectivement incrémentées et décrémentées systématiquement en même temps, au même endroit du programme.

Respect des consignes

L'énoncé s'ouvrait sur un ensemble de consignes clairement détaillées, incluant un encadré en page 2, intitulé par le mot 'Important' en gras. Dans cette première partie d'énoncé réservée aux conseils et consignes divers, était aussi rappelé la différence entre fonction et procédure : les fonctions renvoient une valeur et les procédures ne renvoient rien. Dans toutes les questions, était clairement précisé s'il fallait présenter une fonction ou une procédure. Cependant, dans de nombreux cas, les candidats ont terminé leur programme par return grille quand l'énoncé demandait une procédure.

Plusieurs questions demandaient explicitement de vérifier si le barreau était contre un bord ou en bas de la grille, avant un mouvement, ce qui a souvent été négligé. Les consignes décrivaient une grille dont la ligne 0 était en bas. Certains ont mis la ligne 0 en haut et ont tout inversé dans tout ou partie des 3 premières parties. Certains encore ont inventé des consignes, comme afficher la grille après chaque mouvement du barreau dans les premières questions.

Calcul de complexité

Il y a manifestement une incompréhension profonde du calcul de la complexité d'un algorithme. Plusieurs copies ont exprimé la complexité en fonction d'un paramètre 'n' jamais explicité, alors que le problème s'exprimait en fonction de la longueur et la largeur de la grille. Le jury tient à rappeler que la notion de complexité d'un algorithme est une connaissance de base en informatique qui doit être maîtrisée.

Erreurs diverses

La partie I de l'énoncé commençait par une description claire de la grille et de la manière d'accéder à la valeur d'une case. Cependant, dans de nombreuses copies, il y a eu des interversions entre les numéros de ligne et de colonne. Les correcteurs ont rencontré d'innombrables erreurs de limites. En particulier, si la case au bas du barreau était en colonne y, alors celle du haut était en case y+k-1 et non y-k.

Certains candidats semblent avoir de grandes difficultés à sortir du schéma stéréotypé pour les boucles for : for k in range(n) et, quand il faut utiliser for i in range (k), utilisent k à la place de i dans le contenu de la boucle. Cela serait passé pour de l'inattention, si cette erreur n'était pas apparue aussi souvent dans les copies.

Commentaires détaillés

Pour chaque question, un tableau récapitule les taux de réussite avec les conventions suivantes :

- 0 signifie qu'aucun point n'a été donné pour la question (question non traitée ou totalement fausse) ;
- < 0.5 signifie que moins de la moitié des points ont été donnés ;
- ≥ 0.5 signifie que plus de la moitié des points ont été donnés ;
- 1 signifie que la totalité des points ont été donnés.

Partie 1

Cette partie permettait au candidat de se familiariser le candidat avec les grilles manipulées, *via* la réalisation de deux fonctions simples (initialisation et affichage de la grille). Malgré la simplicité de ces questions, il est frappant de constater le nombre d'erreurs élémentaires commises par les candidats.

Question 1

0		< 0.5		≥ 0.5		1		Total	
52	10%	0	0%	48	10%	391	80%	491	100%

Cette question portait sur la création d'une grille vide. Il fallait bien sur initialiser toutes les cases à VIDE et non à none. La principale erreur rencontrée a été l'interversion entre hauteur et largeur dans la création de la grille. Le préambule demandait d'utiliser, pour la création de listes, la commande [k for i in range (n)], ce qui a été très peu appliqué. La copie d'une colonne dans les différentes case d'une liste ne fonctionne pas en python. Ce dernier point fait cependant partie des subtilités du langage python dont la maîtrise n'est pas nécessairement attendue à ce niveau.

Question 2

0		< 0.5		≥ 0.5		1		Total	
196	40%	0	0%	106	22%	189	38%	491	100%

Cette question portait sur l'affichage de la grille en utilisant trois procédures d'affichage explicitées dans l'énoncé. Il y a eu de nombreuses erreurs pour cette question pourtant simple. Il fallait afficher les lignes de la dernière à la première, comme rappelé dans l'énoncé de la question. Beaucoup

de copies n'ont pas pris en compte cette remarque. D'autres ont vainement essayé. Cela a donné parfois des inversions des colonnes au lieu des lignes.

Dans certaines copies, les lignes étaient affichées en commençant par la ligne numéro colonne, ce qui correspond en python à la ligne 0. Une autre erreur fréquente a été l'affichage colonne par colonne au lieu de ligne par ligne, ce qui correspond à renverser la grille dans l'affichage. Certains n'ont pas placé nouvelleLigne() dans la bonne boucle, affichant une seule valeur par ligne. Enfin, pour chaque case, certains ont utilisé des tests if pour chacunes des couleurs possibles de la case, alors que seule la valeur VIDE nécessitait une exception.

Partie 2

Dans cette partie, il est demandé aux candidats d'implémenter les mouvements du barreau dans la grille de jeu. Les questions sont simples dans l'ensemble; il fallait juste faire écrire les algorithmes avec soin.

Question 3

0		< 0.5		≥ 0.5		1		Total	
125	25%	6	2%	124	25%	236	48%	491	100%

Il fallait ici détecter si une colonne de la grille pouvait recevoir un nouveau barreau. Cette question a été globalement bien traitée malgré des erreurs dans les bornes sur les indices. Il fallait examiner les valeurs sur les lignes hauteur-k <= x <= hauteur-1. Il était clairement indiqué qu'on ne pouvait supposer la grille 'tassée', et donc on ne pouvait pas vérifier uniquement la ligne hauteur-k, ce qui a pourtant été la solution proposée par quelques copies.

Dans certaines copies, le programme vérifie simplement que toutes ces lignes étaient intégralement vides, du fait d'erreurs de conception de l'algorithme. Il fallait soit une variable booléenne ou entière intermédiaire, soit une fonction intermédiaire pour arriver au résultat. Cette question a particulièrement mis en évidence la compréhension des boucles utilisées.

Question 4

0		< 0.5		≥ 0.5		1		Total	
86	18%	24	4%	155	32%	226	46%	491	100%

Dans cette question, il fallait faire descendre le barreau, quand c'est possible. Pour vérifier si le barreau peut descendre, il fallait tester y>0 (précisé explicitement dans le préambule de la question), ce qui a souvent été négligé. La vérification que la case en question (si elle existait) était vide a plus souvent été effectué. Il fallait également mettre à VIDE la case grille[x][y+k-1], ce qui a été parfois oublié. Certains ont modifié la valeur de y à la fin du programme, la considérant comme une variable globale. Ce n'était pas demandé, mais ce n'était pas illogique.

Question 5

0		< 0.5		≥ 0.5		1		Total	
61	12%	18	4%	181	37%	231	47%	491	100%

Dans cette question, il fallait déplacer un barreau en fonction d'un paramètre direction, donné en argument, si ce mouvement est possible. Pour vérifier si le mouvement est possible, il ne fallait, d'après l'énoncé, pas de déplacement à droite depuis le bord droit, pas de déplacement à gauche contre le bord gauche, ou pas de déplacement si toutes les cases d'arrivée ne sont pas vides.

Certains n'ont testé que la case du bas pour les cases d'arrivée, et beaucoup ont négligé les bords de la grille. D'autres encore ont interdit un mouvement à droite depuis le bord gauche ou réciproquement. Certains n'ont pas mis à VIDE les anciennes positions du barreau, ce qui revenait à dupliquer le barreau dans une direction donnée. Il y a eu de nouveau des erreurs de limite pour l'intervalle de cases à déplacer.

Question 6

0		< 0.5		≥ 0.5		1		Total	
210	43%	0	0%	51	10%	230	47%	491	100%

Dans cette question, il fallait permuter les cases du barreau du bas vers le haut. Cela ne nécessitait pas de vérification, mais une exécution dans un ordre précis pour ne pas permuter dans le mauvais sens. Les deux algorithmes proposés par les candidats ont consisté soit à inverser les valeurs des cases deux par deux du haut vers le bas, soit à garder la case du haut en mémoire, pour ensuite copier la valeur de chaque case vers le haut en commençant par le haut.

Pour ceux qui ont utilisé la seconde méthode, beaucoup ont compris qu'il fallait garder en mémoire la position de la case du haut (en ligne y+k-1), mais il fallait aussi déplacer les valeurs en commençant par le haut, ce qui a échappé à énormément de candidats. Certains ont recopié le barreau entier pour ensuite copier depuis la mémoire vers la grille. Cette méthode évitait de s'inquiéter de l'ordre d'exécution, mais utilisait un espace mémoire inutile.

Question 7

0		< 0.5		≥ 0.5		1		Total	
110	22%	76	15%	241	49%	64	14%	491	100%

Cette question demandait l'implémentation de la descente rapide, soit la descente du barreau jusqu'à la plus basse case possible. Certains ont utilisé la procédure descente, implémentée en question

4, soit un algorithme de complexité O(k*hauteur), contrairement à ce qui était demandé. La gestion du bas de la grille a souvent été négligée. Encore une fois, beaucoup ont rajouté l'hypothèse d'une grille tassée. Certains n'ont pas mis à VIDE les cases libérées.

Partie III

Cette partie s'intéresse à la détection des alignements et au calcul du score, ce qui demandait un effort algorithmique un peu plus avancé que dans les 2 premières parties.

Question 8

0		< 0.5		≥ 0.5		1		Total	
269	55%	0	0%	61	12%	161	33%	491	100%

Cette question demandait un simple calcul de score sur une grille donnée en exemple. Les correcteurs y ont pourtant trouvé d'innombrables erreurs, depuis de simples erreurs de recopie à des incompréhensions totales du mode de calcul. Certains ont effectué des calculs de score avec un barreau encore à mi-hauteur de la grille. La partie III commençait pourtant par 'Lorsqu'un barreau ne peut plus descendre ...'.

Question 9

0		< 0.5		≥ 0.5		1		Total	
125	25%	50	10%	265	54%	51	11%	491	100%

La question 9 était la première question réclamant un programme évolué basé sur un algorithme complexe. Il fallait ici calculer le score correspondant à un tableau donné en argument, ainsi qu'un second tableau reprenant les valeurs du premier, sauf pour les cases liées à un alignement, qu'il fallait mettre à VIDE. La majorité des candidats ont montré une bonne intuition de l'algorithme à utiliser, mais la mise en oeuvre fut laborieuse. Du fait de la difficulté de la question, des points ont été accordés pour cette intuition. Beaucoup de copies ont négligé le premier ou le dernier intervalle, voire même les deux. Le calcul commençait et terminait aux premier et dernier changements de couleurs. Très peu ont pensé à vérifier si l'intervalle en cours de calcul ne correspondait pas à des cases à VIDE, ce qui revenait à accorder un score maximal à la grille vide, par exemple.

Certains ont calculé un tableau des indices de changement de valeurs, ce qui était une bonne idée, à condition d'y rajouter le début et la fin du tableau rangee, ou à les prendre en compte dans le calcul du score et du tableau marking. D'autres ont simplement regardé une fenêtre glissante de 3 cases du tableau pour vérifier si les valeurs étaient identiques. C'était effectivement une bonne idée qui permettait un calcul simplifié du score et du tableau marking. Cependant, l'énoncé interdisait explicitement de regarder plusieurs fois un même élément, ce qui exclut de fait cette méthode. Au final,

du fait de toutes ces erreurs, le tableau marking n'avait pas forcément la même longueur que le tableau rangee.

Question 10

0		< 0.5		≥ 0.5		1		Total	
184	37%	70	14%	106	22%	131	27%	491	100%

Dans cette question, il fallait produire un tableau à partir d'une grille, en partant d'une position donnée et dans une direction donnée. L'énoncé précisait bien que la direction était donnée par des paramètres dx et dy, à valeurs dans {-1,0,1}. Cela signifie qu'il fallait exclure le cas dx = 0 et dy = 0, sous peine d'obtenir une boucle infinie. Cette subtilité n'a presque jamais été prise en compte par les candidats.

Cependant, certains ont recopié grille dans g, ce qui montre une incompréhension du fonctionnement global de la méthode de calcul du score. D'autres ont mis toute la rangée étudiée à VIDE dans g, sans étudier la valeur du tableau marking. L'énoncé précisait pourtant qu'il ne fallait mettre a jour dans g que les cases appartenant à un alignement. Pour remplir le tableau intermédiaire rangee, certains ont vérifié le bord droit et le haut de la grille, mais pas le bord gauche et le bas (x+k*dx >= 0 et y+k*dy >= 0). Quelques rares candidats ont pensé à calculer au préalable le nombre de cases du tableau rangee à construire, pour éviter de vérifier, à chaque boucle du while, quatre comparaisons différentes. Enfin, pour modifier g en fonction de marking, plusieurs candidats ont eu besoin de 3 variables différentes, pour la ligne, la colonne, et la position dans marking, ce qui était parfaitement redondant.

Question 11

0		< 0.5		≥ 0.5		1		Total	
330	67%	15	3%	48	10%	98	20%	491	100%

Dans cette question, il fallait calculer le score des alignements d'une grille, et produire une seconde grille dans laquelle étaient mis à VIDE les cases appartenant à au moins un alignement. Cette question a été beaucoup moins traitée que les précédentes. Il fallait appliquer la fonction scoreRangee sur un ensemble bien choisi de valeurs de x, y, dx et dy. Beaucoup de candidats l'ont simplement lancé sur toutes les valeurs possibles entre 0 et n-1 et entre -1 et 1 respectivement, ce qui revient à compter plusieurs fois les alignements possibles.

Parmis ceux qui ont compris qu'il ne fallait donner des valeurs de x et y que pour des cases sur deux bords de la grille, certains ont oublié une des deux diagonales ou compté deux fois la diagonale partant de la case (0,0).

Question 12

0		< 0.5		≥ 0.5		1		Total	
353	72%	61	12%	9	2%	68	14%	491	100%

Dans cette question, il fallait modifier une procédure pour tasser la grille. Les candidats ont bien compris qu'il fallait opérer colonne par colonne, mais les méthodes proposées étaient pour l'essentiel, soit erronées, soit de complexité inutilement élevée. Beaucoup de candidats ont parcouru chaque colonne en s'arrêtant lorsqu'ils trouvaient une case vide sous une case non vide pour les intervertir. Cela ne fonctionne pas dans des boucles for. En parcourant de bas en haut, cela descendra d'une seule case les blocs non tassés, et dans l'autre sens, cela ne descendra correctement qu'une seule case de chaque bloc. D'autres ont appelé la fonction descenteRapide sur des barreaux de hauteur 1. L'algorithme fonctionne mais est inutilement lent. Il faut recalculer systématiquement la nouvelle position de la case, qui peut être simplement gardée en mémoire.

Très peu de candidats ont proposé des méthodes pertinentes. Quelques uns ont recopié dans un tableau intermédiaire les valeurs non VIDE rencontrées dans chaque colonne, pour ensuite recopier ce tableau dans la colonne étudiée. Cela fonctionne au prix d'un espace mémoire non nécessaire (ce qui n'a pas été pénalisé). Encore fallait-il ajouter le bon nombre de valeurs VIDE pour le haut de la colonne, ce qui a été source de nouvelles erreurs.

Question 13

0		< 0.5		≥ 0.5		1		Total	
344	70%	0	0%	48	10%	99	20%	491	100%

Il fallait ici calculer le score et la nouvelle grille en utilisant les questions précédentes. Cette question a été plutôt bien comprise par ceux qui l'ont traitée, mais il ne fallait surtout pas chercher à comparer les grilles, comme condition d'arrêt. La comparaison de deux grilles est de complexité élevée, alors que l'on pouvait simplement comparer les scores.

Partie IV

Dans cette partie, on considère une variante du jeu où il s'agit de former des régions unicolores plutôt que des alignements. Il était demandé d'écrire un algorithme d'exploration récursive ainsi que d'analyser le fonctionnement d'un algorithme.

Question 14

0		< 0.5		≥ 0.5		1		Total	
454	92%	18	4%	9	2%	10	2%	491	100%

Dans cette question, Il fallait proposer une fonction récursive calculant la région maximale de même couleur à partir d'une case donnée. Peu de candidats ont traité cette question. Il fallait marquer les cases visitées, en mettant la case à VIDE ou en remplissant une seconde grille de booléens.

Question 15

0		< 0.5		≥ 0.5		1		Total	
481	98%	4	1%	0	0%	6	1%	491	100%

Il fallait ici déterminer si la fonction proposée remplissait son rôle. Cette question — difficile — a été très peu traitée et l'essentiel des réponses étaient fausses.

Partie V

Dans cette partie, il est demandé aux candidats d'écrire quelques requêtes SQL sur une base de données simple.

Question 16

0		< 0.5		≥ 0.5		1		Total	
195	40%	0	0%	45	9%	251	51%	491	100%

Il fallait ici donner une requête SQL renvoyant les données de toutes les parties jouées par un joueur de nom cc, donné en paramètre. Beaucoup de candidats on utilisé cc comme clé primaire, et non comme nom du joueur. Certains ont utilisés plusieurs selects, pour cette requête basique.

Question 17

0		< 0.5		≥ 0.5		1		Total	
298	61%	11	2%	38	8%	144	29%	491	100%

Il fallait ici proposer une requête le rang d'un score donné par rapport aux parties de la base de donnée. Il fallait renvoyer count+1 pour obtenir la bonne valeur, comme indiqué dans la consigne.

Question 18

0		< 0.5		≥ 0.5		1		Total	
260	53%	10	2%	30	6%	191	39%	491	100%

Dans cette question, il fallait trouver le meilleur score de chaque pays. Beaucoup de candidats semblent ignorer la fonction MAX.

Question 19

0		< 0.5		≥ 0.5		1		Total	
402	82%	16	3%	26	5%	47	10%	491	100%

Dans cette question, il fallait trouver le classement d'un joueur, c'est-à-dire sa position dans le classement des meilleurs scores. Pour cela, il fallait utiliser DISTINCT. C'est une question complexe, qui a été peu traitée.