Examen première session 16/12/2021

*Durée : **2h**. Document autorisé : **1 feuille A4 recto-verso**. Il n'est pas demandé d'écrire un `main`.*

1. Déclarer un type `coord` représentant une coordonnée sous la forme d'un entier court signé, et définir une classe `position` composée d'un couple de coordonnées. Cette classe disposera d'un unique constructeur permettant d'initialiser une position à partir de deux coordonnées, de deux accesseurs `x` et `y` aux coordonnées, d'un opérateur de comparaison par égalité (==) et un opérateur de sortie sur flux.
2. Sur un plateau de jeu (que nous écrirons plus tard), il y a différentes entités, chaque entité a un identifiant unique attribué automatiquement. Les 3 types d'entités sont les suivants :
   Joueur. Un joueur a un nom (chaîne de caractères) fixé à la construction.
   Obstacle.
   Bombe. Une bombe peut être dans l'état *décompte* ou dans l'état *explosion*. À la construction, une bombe est dans l'état décompte. Une bombe a aussi un compte à rebours. À la construction, une bombe est créée avec un compte à rebours à 10, cette valeur de 10 est appelée *durée par défaut*. Cette durée par défaut doit pouvoir être modifiée par appel à une méthode `modifierduree`. Les bombes déjà créées ne seront pas modifiées par un appel à `modifierduree`.
   Écrire une/des classe(s) pour représenter ces différents types d'entités ainsi que les constructeurs. **Attention** Il n'est pas demandé de représenter la `position` d'une entité. La classe `position` sera utilisé plus tard dans l'exercice.
3. Écrire une méthode `symbole` retournant le symbole d'une entité : pour un joueur, il s'agit de l'initiale de son nom, pour un obstacle : #, pour une bombe : `B` si elle est dans l'état *décompte* ou `E` dans l'état *explosion*.
4. Écrire un opérateur de sortie sur flux sortant les informations sur une entité (symbole, nom du joueur, compte à rebours).
5. Écrire une méthode `decompte` pour une bombe, qui fait passer la bombe dans l'état suivant. Les changements d'état sont les suivants :
   Dans l'état *décompte*. Si le compte à rebours est supérieur à 0, il est décrémenté. S'il est égal à 0, la bombe passe dans l'état *explosion* et le compte à rebours est mis à la valeur *durée par défaut*.
   Dans l'état *explosion*. Si le compte à rebours est supérieur à 0, il est décrémenté.
   Quand une bombe dans l'état *explosion* a un compte à rebours à 0, elle a fini d'exploser, la méthode retourne *vrai* pour signaler que l'objet a fini son existence, dans tous les autres cas, retourne *faux*.
6. Écrire une classe `plateau` représentant un plateau de jeu contenant tous types d'entités. La taille du plateau sera fournie à la construction (et fixe) sous la forme d'une `position`. Vous ne modifierez pas les classes précédentes pour cela (notamment, vous n'ajouterez pas un attribut `position` aux entités). Il n'y aura jamais deux entités à la même position. Vous choisirez une structure de données permettant d'accéder à l'entité située à une `position` donnée en temps constant (c'est-à-dire aucun parcours d'une structure de donnée pour la méthode `recherche` de la question 8).
7. Écrire une méthode `valide` prenant comme argument une `position` et retournant *vrai* si la `position` est valide sur le plateau, *faux* sinon.
8. Écrire une méthode `recherche` prenant comme argument une `position` et permettant d'accéder à l'entité située à cette `position`. Cette méthode lèvera une exception en cas de position invalide. Penser au cas où il n'y a aucune entité à la position (valide) passée. Inutile de définir un nouveau type exception, vous pouvez utiliser un type fourni par la bibliothèque standard comme `std::invalid_argument` dont le constructeur prend comme argument une chaîne.
9. Écrire une méthode `ajouter` permettant d'ajouter une entité à une `position` donnée. Si une entité se trouve déjà à cette `position`, elle est supprimée et remplacée par celle qui est ajoutée. Si la `position` n'est pas valide, cette méthode lèvera une exception.
10. Faire en sorte que la copie d'un `plateau` provoque une erreur de compilation facile à comprendre.
11. Écrire une méthode `idmax` retournant l'identifiant d'entité le plus grand parmi les entités du plateau.

12. Faire en sorte qu'on puisse utiliser un `plateau` comme une sorte de tableau dont les indices seraient des identifiants d'entités. Par exemple, soit `p` un `plateau`, `p[5]` permet d'accéder à l'entité d'identifiant 5. Une exception sera levée en cas d'entité introuvable.

13. Écrire une **fonction** `nbjoueurs` (**pas** une méthode) qui utilise (uniquement) les deux méthodes précédentes pour retourner le nombre de joueurs d'un `plateau`.

14. Écrire un opérateur de sortie sur flux faisant une sortie de ce type (sur cet exemple, `1` et `2` sont les initiales des 2 joueurs) :

```
###############
#             #
#    1        #
#  B  2       #
#             #
#  #          #
###############
```

15. Écrire une méthode `deplacerjoueur` prenant comme paramètres deux positions `src` et `dst`, et déplaçant le joueur situé en `src` à la position `dst`. Un joueur peut se déplacer sur une position libre ou contenant une bombe (dans ce cas, la bombe se trouvant en `dst` sera supprimée). Tous les autres déplacements sont interdits. Cette méthode retournera *vrai* si le déplacement a été effectué, *faux* sinon (en cas de position invalide ou déplacement interdit).

16. Écrire une méthode `explosion`, prenant comme argument une `position p`, une largeur d'explosion `l` et un ensemble de `position r` et rajoutant à `r` les positions d'explosion accessibles à partir d'une bombe explosant à la position `p`, avec une largeur maximale de `l`. Une bombe explose en « + », et l'explosion est arrêtée par un obstacle. Par exemple, avec une position signalée par `E` et une largeur de 4, les positions devant être stockées dans `r` sont celles signalées par un « * » ici : (on ne demande pas d'afficher ces coordonnées, simplement les stocker dans `r`)

```
###############
#             #
#  #          #
#**E****      #
#  *          #
#  *          #
###############
```

17. Écrire une fenêtre d'interface avec Qt permettant de saisir un couple de coordonnées, un type d'entité (joueur, bombe, obstacle), une « information supplémentaire », et rajoutant à un plateau l'entité correspondante. Dans le cas d'un joueur, l'information supplémentaire sera le nom du joueur, et dans le cas d'une bombe, la largeur de la bombe. Si l'ajout est impossible (position invalide notamment), un message d'erreur sera affiché dans un `QLabel` prévu à cet effet.

18. Écrire une méthode `toursuivant` permettant de calculer l'état suivant du plateau : le décompte des bombes est appelé, les bombes ayant fini d'exploser sont supprimées, les positions des explosions sont calculés, les joueurs qui sont dans une explosion sont supprimés.

Barème donné à titre indicatif

Question	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Points	1,5	3	1	1,5	1	1,5	0,5	2	1	1	0,5	1	1,5	1,5	2	2	3	3
Total	1,5	4,5	5,5	7	8	9,5	10	12	13	14	14,5	15,5	17	18,5	20,5	22,5	25,5	28,5