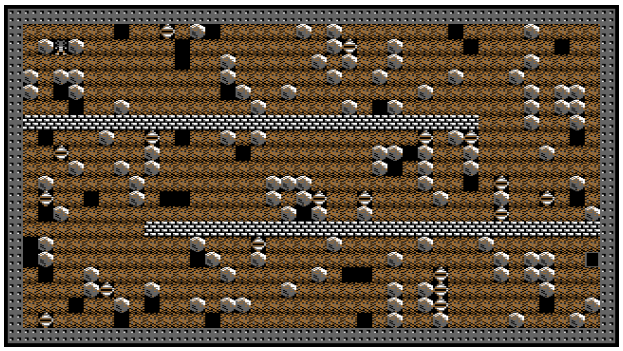
Projet : Boulder Dash

Le but du projet est de réaliser un jeu de Boulder Dash. Le joueur (historiquement nommé *Rockford*), doit creuser la terre pour se frayer un chemin vers la sortie d'une grotte en ramassant des diamants. Il doit faire attention à ne pas se faire écraser par un rocher ou un diamant ni se faire toucher par d'éventuels ennemis. Voici une version en ligne à laquelle vous pouvez jouer pour vous faire une idée du fonctionnement du jeu : `http://lortschi.de/games/bd/`.



Le jeu se joue sur une grille. Initialement, chaque case peut être soit vide, soit contenir l'un des objets suivants :

- le personnage *Rockford* (R),
- de la terre (G) que l'on peut creuser,
- un mur (W) infranchissable,
- un rocher (B),
- un diamant (D),
- la sortie (E).

Rockford peut se déplacer naturellement sur les cases vides dans toutes les directions et ne peut pas traverser les murs. Il peut ramasser les diamants (qui disparaissent alors de leur case) et pousser horizontalement les rochers dans les cases vides. Il peut creuser les cases de terre qui deviennent alors des cases vides.

Lorsque l'on crée des cases vides, les rochers peuvent tomber. Cela arrive uniquement dans les deux cas de figure suivants :

- Si un rocher est au dessus d'une case vide, il tombe dans la case vide.
- Si, à un moment donné, deux rochers sont empilés l'un sur l'autre, et que les deux cases à leur droite (resp. à leur gauche) sont vides, alors le rocher du dessus tombe dans la case vide à droite (resp. à gauche) en dessous.

Ces éboulements peuvent avoir lieu en cascade (lorsqu'un éboulement crée à nouveau une configuration d'éboulement). Et si *Rockford* se trouve dans la case juste en dessous d'une case où un rocher tombe, il est écrasé et la partie est perdue. Pour gagner la partie, il doit ramasser un certain nombre de diamants dans le temps imparti.

Des règles plus précises vont être données dans la suite des consignes.

Votre programme devra pouvoir **lire un niveau depuis un fichier** dans le format suivant. Sur la première ligne sont indiqués le temps imparti en secondes et le nombre de diamants à ramasser pour pouvoir sortir. Ensuite, le plan du niveau est indiqué par une grille de caractères. Chaque caractère correspond à un des types d'objet listés ci-dessus. Les points correspondent à des cases vides.

```
150s 1d
WWWWWWWWWWWW
WGGGGGG.GGDGBW
WGBRBGGGGGG.GW
WGGGGGGGGGG.GW
WBGBBGGGGGGGGW
WGG.BGGGGGGGGW
WGGG.GG.GGGGEW
WWWWWWWWWWWW
```

1 Le programme à réaliser

Vous allez réaliser votre jeu en 3 phases. Chaque phase devra être terminée et fonctionnelle avant de pouvoir passer à la suivante.

1.1 Phase 1 : la base

Dans un premier temps, vous aller créer la base du jeu. Dans cette phase, un niveau de jeu sera codé "en dur", c'est-à-dire directement dans le programme. C'est à vous de choisir la structure de données que vous utiliserez pour cela. Pour l'instant, nous n'allons pas gérer les effets de vitesse de déplacement lors des éboulements de rocher (c'est-à-dire que *Rockford* ne peut pas échapper à un éboulement et que les éboulements en cascade sont effectués avant que l'on puisse déplacer *Rockford*). À la fin de cette phase de développement, on doit pouvoir :

1. lancer le jeu avec un niveau qui contient tous les types d'objets listés ci-dessus ;
2. déplacer *Rockford* sur des cases vides ou en creusant la terre ;
3. pousser les rochers horizontalement et ramasser des diamants ;
4. faire tomber un rocher verticalement lorsque qu'une case vide est créée en-dessous (et gérer les chutes verticales en cascade si nécessaire) ;
5. vérifier si la partie est gagnée (pour la phase 1, il suffit d'atteindre la sortie, on ne se préoccupe ni du nombre de diamants, ni du temps écoulé) ;
6. ajouter la possibilité de faire un "reset", pour améliorer la jouabilité ;
7. ajouter un mode *debug* : on rajoute une touche au choix qui permet d'activer un mode où *Rockford* se déplace de façon aléatoire (en poussant éventuellement des rochers). Si on appuie sur cette touche à nouveau, on peut recommencer à jouer normalement. On doit pouvoir passer en mode *debug* aussi souvent qu'on le souhaite.

Conseil 1 La liste ci-dessus est une bonne indication de l'ordre dans lequel procéder pour écrire votre programme (sauf la dernière, qui a intérêt à être implémentée dès que possible pour servir de test).

Conseil 2 Vous avez fortement intérêt à séparer votre programme en 3 parties principales :
- celle qui contrôle le jeu (la boucle principale du jeu, la gestion des touches, ...) ;
- celle qui gère l'affichage (et notamment la correspondance entre votre structure de données et les cases affichées) ;
- celle qui maintient toutes les données du jeu (emplacement des éléments sur la grille, point de départ, position de *Rockford*, ...)

1.2 Phase 2 : règles avancées et gestion des niveaux

Dans cette phase, vous devez modifier le jeu pour :
- prendre en compte les diamants : ils font gagner des secondes et des points et il faut en avoir ramassé un certain nombre pour que la sortie soit utilisable (elle est visible tout au long de la partie mais ne permet de sortir qu'avec suffisamment de diamants : elle change de couleur quand elle devient utilisable) ;
- prendre en compte le temps : afficher l'écoulement du temps et faire en sorte que la partie soit perdue si le temps imparti est écoulé.
- prendre en compte les éboulements latéraux (toujours sans effets de vitesse).

À la fin de cette phase de développement, on doit pouvoir également :

— créer un **niveau aléatoire**. Nous vous conseillons de mettre beaucoup de cases de terre par rapport aux autres objets pour avoir de bonnes chances que le niveau soit faisable. Vous pouvez essayer de garantir que la sortie est accessible, mais c'est plus difficile (dans ce cas, discutez-en avec votre chargé de TP) ;
— **lire** un niveau à partir d'un **fichier** formaté comme indiqué au début du sujet. Le nom du fichier doit pouvoir être passé en argument de la ligne de commande ;
— **sauvegarder** la partie en cours et la **recharger** ;
— garder les meilleurs scores des parties précédentes par niveau.

Vous devrez rajouter vos propres niveaux pour pouvoir faire plus de tests. Vous pouvez vous inspirer de ceux qui se trouvent ici : `https://www.boulder-dash.nl/` et en trouver d'autres en ligne (dans tous les cas, vous devez indiquer les sources de ces niveaux dans le jeu).

1.3 Phase 3 : extensions

Dans cette phase, vous réaliserez au moins deux extension (une de type 1 et une de type 2) :

Extension 1.a Rajouter la gestion de la vitesse de déplacement (c'est-à-dire faire en sorte que *Rockford* puisse échapper à un éboulement s'il se déplace assez vite).

Extension 1.b Rajouter des ennemis (des lucioles). *Rockford* meurt s'il en touche une. Vous pouvez les faire exploser en faisant tomber un rocher dessus (ça détruit le contenu de la case où elle se trouve ainsi que le contenu des 8 cases adjacentes). Une luciole se déplace à vitesse constante le long du bord de la zone vide où elle se trouve, dans le sens des aiguilles d'une montre.

Extension 2.a Rajouter un menu de sélection des niveaux et/ou du type de niveau (aléatoire ou fichier). On doit néanmoins toujours avoir une version du jeu qui peut lire un fichier de niveau en argument en ligne de commande et le lancer directement avec ce niveau.

Extension 2.b Construire un éditeur de niveau graphique qui permettra de construire à la souris des fichiers de niveaux, en choisissant le différents éléments à y placer dans un menu.

Notation et extensions

Vous devez impérativement réaliser les deux premières phases du *Boulder Dash*. Vous serez alors notés sur 16. Vous allez ensuite réaliser des extensions. Attention, vous devez **d'abord** avoir une version du jeu sans extension qui fonctionne parfaitement (pensez à faire des tests régulièrement et des sauvegardes des versions qui fonctionnent).
Pour être noté sur 20, vous devez réaliser au moins une des extensions numéro 1 ET une des extensions numéro 2. Si vous souhaitez proposer de nouvelles extensions, parlez-en à votre chargé de TP avant.

2 Consignes de rendu

Vous devez rendre une version du projet à la fin de chacune des 3 phases

— La date limite pour le premier rendu est le **lundi 11 novembre 2019 à 23h55**. Passé ce délai, la note attribuée à votre projet sera 0. Les séances de TP de la semaine suivante seront consacrées à la vérification de votre rendu et à amorcer la seconde partie.
Remarque : vous ne devez pas commencer la phase 2 avant d'avoir parfaitement rempli l'objectif de la phase 1.

— La date limite pour le second rendu est le **15 décembre 2019 à 23h55**.
— La date limite pour le troisième rendu est le **12 janvier 2019 à 23h55**.
Des mini-soutenances seront organisées quelques jours après le rendu, pendant lesquelles vous ferez une démonstration sur une machine des salles de TP et serez interrogés sur le projet (les choix techniques/algorithmiques que vous avez faits, les difficultés rencontrées, l'organisation de votre travail, ...). Les contributions de chaque participant seront évaluées, et il est donc possible que tous les participants d'un même groupe n'aient pas la même note.

Contraintes

— Ce projet est à faire en binôme (s'il y a besoin de faire une exception, contactez les enseignants). Vous devrez sélectionner votre binôme sur e-learning dans la semaine suivant la publication du sujet.
— **Vous DEVEZ utiliser** `upemtk`. L'utilisation de modules autres que les modules standards de Python est interdite ; en cas de doute, n'hésitez pas à poser la question.
— **Votre programme devra impérativement s'exécuter sans problème sur les machines de l'IUT.** Prévoyez donc bien de le tester sur ces machines en plus de la vôtre et d'adapter ce qui doit l'être dans ce cas (par exemple, les vitesses de déplacement). Il sera donc utile de permettre à l'utilisateur de préciser certaines options auxquelles vous attribuez des valeurs par défaut plutôt que de devoir modifier le code à chaque exécution.

Le format de chaque rendu est une archive à déposer sur la plate-forme e-learning et contenant :

1. **les sources de votre projet**, commentées de manière pertinente (quand le code s'y prête, pensez à écrire les doctests). De plus :
   — les fonctions doivent avoir une longueur raisonnable ; n'hésitez pas à morceler votre programme en fonctions pour y parvenir ;
   — plus généralement, le code doit être facile à comprendre : utilisez des noms de variables parlants, limitez le nombre de tests, et veillez à simplifier vos conditions ;
   — quand cela se justifie, regroupez les fonctions par thème dans un même module ;
   — chaque fonction et chaque module doit posséder sa *docstring* associée, dans laquelle vous expliquerez le fonctionnement de votre fonction et ce que sont les paramètres ainsi que les hypothèses faites à leur sujet.
2. un **fichier texte** `README.txt` expliquant :
   — les extensions qui ont été implémentées,
   — l'organisation du programme,
   — les choix techniques,
   — les éventuels problèmes rencontrés.
   Vous pouvez y ajouter tous les commentaires ou remarques que vous jugez nécessaires à la compréhension de votre code.

Si le code ne s'exécute pas, la note de votre projet sera 0. Vous perdrez des points si les consignes ne sont pas respectées, et il va sans dire que si deux projets trop similaires sont rendus par 2 binômes différents, la note de ces projets sera 0.