Lycée des GRAVES
Mise à jour 06/04/22

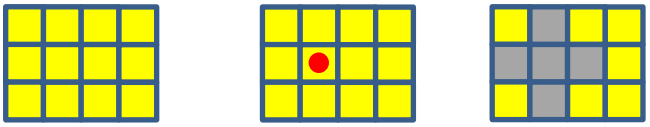
Light_out

1. Présentation du projet

Le jeu light_out en console

1.1. Les règles du jeu

On se place dans un rectangle de longueur L et de hauteur H, possédant N = L x H cases. Ces cases peuvent être soit éteintes, soit allumées. On a la règle d'évolution suivante : lorsque l'on appuie sur une case, celle-ci change d'état ainsi que ses quatre voisines dans les directions sud, est, nord, ouest, du moins celles qui existent dans les limites du rectangle. Les cellules du coin n'ont que deux voisines, et celles de la bordure sauf les coins en ont trois. Au départ toutes les cases, ou une partie d'entre elles sont allumées. Le jeu consiste à trouver sur quelles cases appuyer pour finir en ayant toutes les cases éteintes.

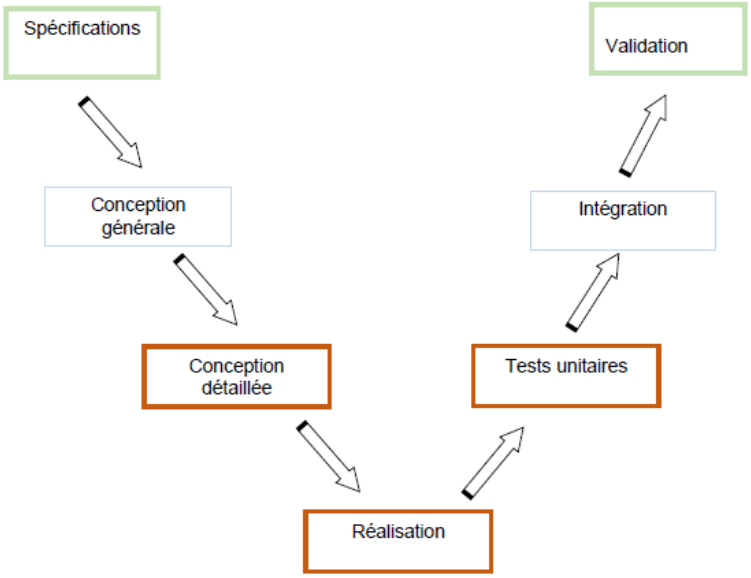


1.2. Le jeu simplifié

- Les case sont notées par deux états
- Phase d'initialisation : On demande au joueur les dimensions du plateau de jeu et un nombre de coups maximum.
- Phase de jeu : Le joueur choisi une case dans la console par ses coordonnées. L'écran est rafraichi.
- Phase de fin : Le jeu se termine soit par une grille gagnante soit par un nombre de coups maximum.

2. Conduite du projet

La conduite de projet suivra la description ci-dessous



Luc.vincent@ac-bordeaux.fr

2.1. Spécifications du jeu

- On choisit d'afficher des 0 ou des 1 dans la console.
- Le dialogue Joueur / Ordinateur s'effectue grâce au clavier et la console.
- On limite le jeu à une grille de 4 à 64 cases.
- Le nombre de coups maximum est de 100.
- Le programme s'assure de la validité des choix de l'utilisateur.

2.2. Conception générale

On commence par construire un algorithme général du jeu

```
# Initialiser le jeu
# créer la structure de donnée représentant la grille
# Afficher la grille
# Tant que la partie n'est pas gagnée et que le nombre de coup maximum n'est pas atteint :
        # Interroger le joueur
        # Calculer l'évolution de la grille
        # Afficher la nouvelle grille.
        # Compter les coups
# Afficher un message de fin
```

On voit ici apparaître une décomposition fonctionnelle. Chaque fonction peut être écrite par un concepteur différent.

Il est nécessaire avant tout que les concepteurs choisissent les structures de données adaptées pour les différentes informations.

☞ Choisir une structure de donnée pour la grille
C'est un tableau d'entiers `board` de dimension 2, de longueur `width` et de hauteur `height` définies dès la phase d'initialisation.

☞ Choisir une structure de donnée pour le nombre de cases limite et le maximum
Une constante de type entier BOX_MIN, BOX_MAX, MAX_TRIALS

☞ Choisir une structure de donnée pour le nombre d'essais réalisés et son maximum choisi
Une variable de type entier `trials`
Une variable de type entier `trials_max`
Une variable de type booléen `playing`

2.3. Conception détaillée

Il s'agit de donner les signatures des fonctions nécessaires. Il sera peut-être nécessaire d'écrire d'autres fonctions *internes* pour exécuter certaines tâches, rendre le code plus lisible…

2.4. Réalisation

Il s'agit maintenant de coder les fonctions envisagées. Pour chaque fonction, le concepteur crée un programme, documente la docString et valide sa fonction avec un jeu de test.

Luc.vincent@ac-bordeaux.fr

```
def initialise(MAX_TRIALS:int, BOX_MAX:int, BOX_MIN:int)->list:
    '''
    MAX_TRIALS: int  Nombre d'essais maxi acceptable
    BOX_MAX : int Nombre maxi de case
    BOX_MIN : int Nombre mini de case
    CU : Valeurs limite arguments 100, 64, 4"
    except : assertionError
    answer : list of int [width, heigth, trials]
    '''
```

```
def play(width:int, height:int)->list:
    '''
    width: int la largeur de la grille de jeu
    height: int la hauteur de la grille de jeu
    CU: 4 < width * height < 64
    except : assertionError
    answer : list of int [abcissa, ordinate]
    0 <= abcissa <= width and 0 <= ordinate <= height
    '''
```

```
def evolution(position:list, board:list):
    '''
    postion:list[int, int] la position jouée abcisse, ordonnée
    board: list [[], []] la platine de jeu
    modifie board en place
    '''
```

```
def somme(board:list)->bool:
    '''
    renvoyer faux si la somme de la grille est nulle
    '''
```

```
def display(board:list):
    '''
    afficher la grille de jeu à l'écran
    board: une grille
    CU: 4 à 64 cases des 0 ou des 1
    '''
```

3. *Intégration*

Les travaux des différents concepteurs sont réunis. Le programme light_out.py peut être implanté en important les travaux réalisés.

4. *Validation du projet*

Par un jeu de test permettant la validation du projet. On peut proposer quelques parties type.

Luc.vincent@ac-bordeaux.fr

5. Travail demandé

Chaque groupe devrait réaliser un document présentant :

- Les spécifications
- La conception générale,
- La conception détaillée,
- Le partage des tâches entre les membres du groupe.

Ce travail a déjà été réalisé dans ce projet. Vous devrez vous en inspirer dans vos prochains projets.

Chaque groupe devra produire un dossier compressé contenant les fichiers du programme :

- Les codes des programmes devront être commentés.
- Les fonctions auront une *docstring* détaillée
- Un jeu de tests sera construit pour chaque fonction
- Les préconditions et les post conditions doivent être précisées

Chaque module commencera toujours par les lignes

```
# Author(s) name (Individual, Team or corporation)
# Date
# Title of program/source code
# Code version
# Type (e.g. main program, module, source code)
# Web address or publisher (e.g. program publisher, URL)
```

6. Compétences évaluées

- ✗ Analyser et modéliser un problème
- ✓ Décomposer un problème en sous problèmes
- ✓ Concevoir des solutions algorithmiques
- ✗ Mobiliser les concepts et les technologies
- ✓ Traduire un algorithme dans un langage de programmation
- ✓ Développer des capacités d'abstraction et de généralisation