Lycée des GRAVES
Mise à jour 06/12/22

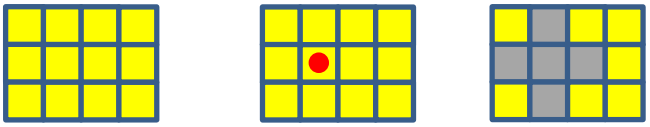
Light_out

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.

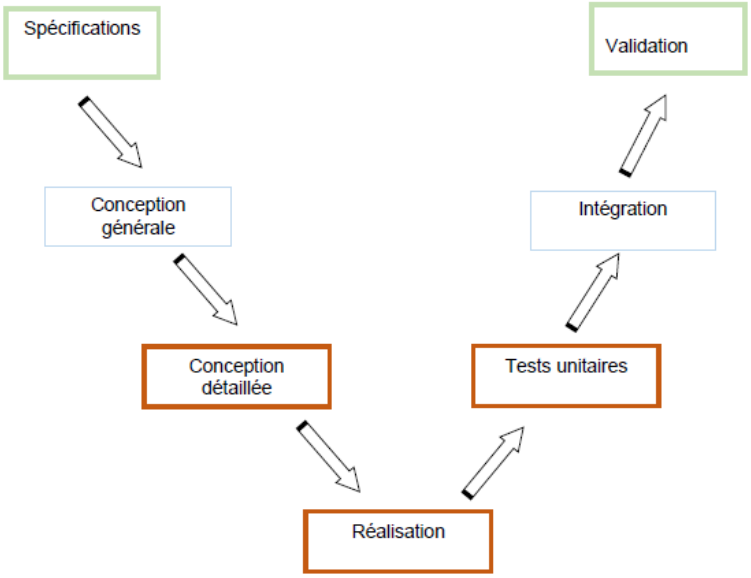


1.2. Le jeu simplifié

- Les case sont notées par ……. états
- Phase d'initialisation : On demande au joueur …
  …
- Phase de jeu : Le joueur choisi …
- Phase de fin : Le jeu se termine soit …

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

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

Chaque case peut être modélisée par un booléen ou par un entier (0/1).
La grille peut être un tableau de tableaux, un simple tableau où toutes les cases sont côte à côte, ou même un tableau de tableaux avec 2 lignes et 2 colonnes supplémentaires pour éviter de traiter les cas de bord et d'angles.
Choix retenu :
C'est un ...

- Choisir une structure de donnée pour le nombre de cases limite et le maximum

Une constante de type ..., BOX_MIN, ..., ... Penser à importer les noms dans chaque module !

- Choisir une structure de donnée pour le nombre d'essais réalisés et son maximum choisi

Une variable de type ..... `trials`
Une variable de type .... `trials_max`
Une variable de type .... `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...
Il vous faudra scinder le code en fonctions réparties dans des modules.

Luc.vincent@ac-bordeaux.fr

2.4. Réalisation

Il s'agit maintenant de coder les fonctions envisagées dans des modules indépendants. Pour chaque fonction, le concepteur crée un programme, documente la docString et valide sa fonction avec un jeu de test.
Voici par exemple ce que vous pourriez obtenir :

```
def initialise()→...:
    '''
    Renvoyer des valeurs valides choisies par l'utilisateur
    CU : Valeurs limite renvoyées MAX_TRIALS, BOX_MAX, MAX_TRIALS"
    except : assertionError
    answer : list of int [width, heigth, trials] choisis par l'utilisateur
    '''
```

```
def play(width:int, ...)→...:
    '''
    Renvoyer une liste abcisse, ordonnee de la case jouée dans la limite des dimensions de la grille de jeu
    width: int la largeur de la grille de jeu
    height: int la hauteur de la grille de jeu
    CU: BOX_MIN < width * height < BOX_MAX
    except : ValueError
    answer : list of int [abcissa, ordinate] player choice
    0 <= abcissa <= width and 0 <= ordinate <= height
    '''
```

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

```
def display(...):
    '''
    Afficher la grille de jeu à l'écran
    board: une grille
    CU: BOX_MIN à BOX_MAX cases avec 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. Le programme light_out.py sera le point d'entrée, où sera codé l'algorithme général du jeu.

4. Validation du projet

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

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 partiellement 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