Présentation de la conduite de projet.

Table des matières

1 Présentation de la conduite du projet

1.1 Étapes et niveaux de conception

Il existe diverses méthodes de conception et de conduite de projet. En NSI, pour commencer simplement nous pouvons utiliser la méthode dite en V.

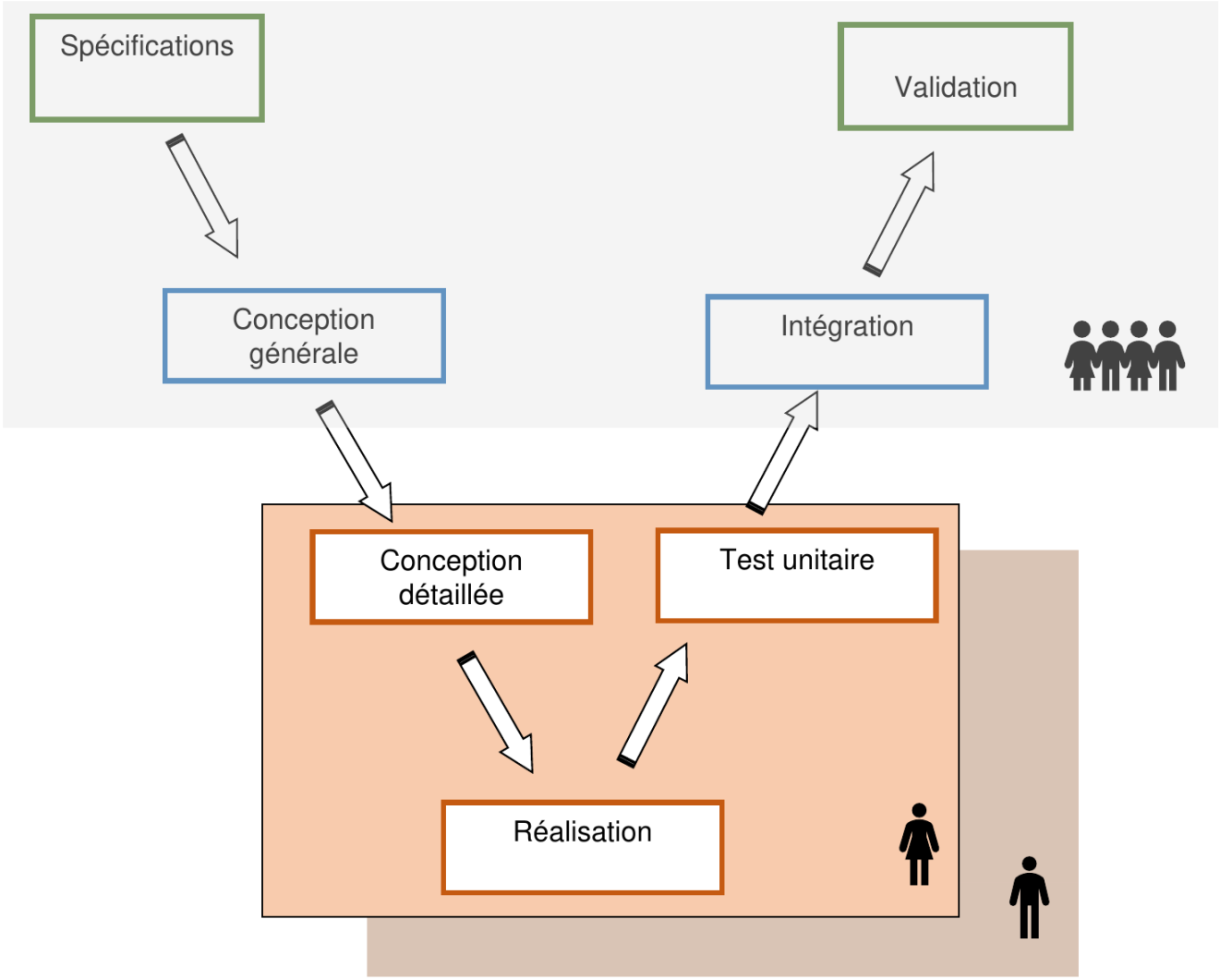
Cette méthode se décompose en deux phases :

- Phase 1 : conception
  - du problème général vers le code élémentaire
  - définir le problème ;
  - décomposition d'un problème en sous-problèmes ;
  - décomposition de fonctions en sous-fonctions.
- Phase 2 : tests et validation
  - tests des codes élémentaires vers les tests des fonctions de haut niveau ;
  - validation du fonctionnement de l'ensemble

Illustration des étapes de la conduite de projet

On remarquer qu'il y a :

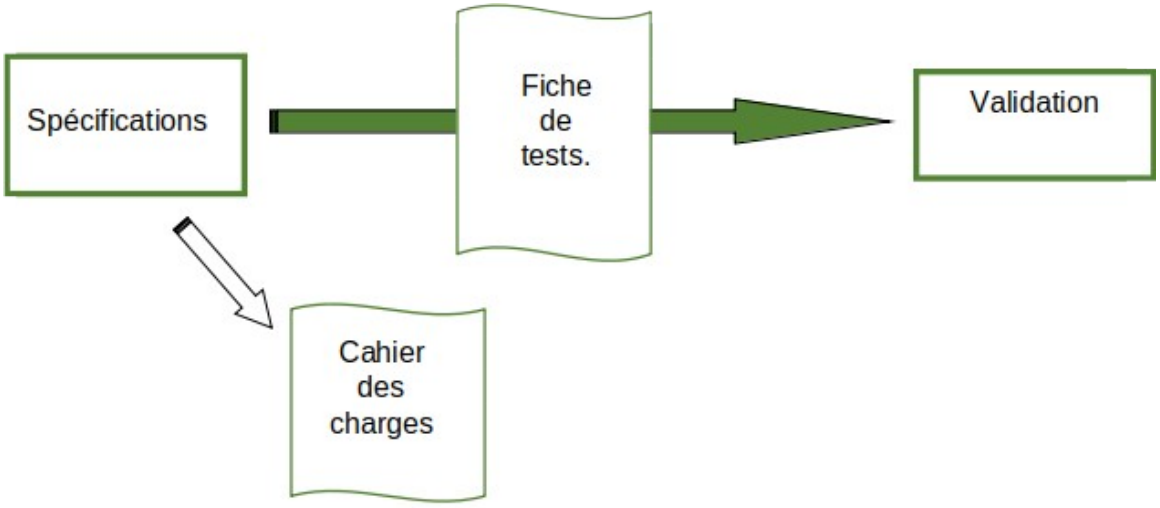
- des étapes de travail en groupe ;
- des étapes de travail individuel



1.2 Niveau Spécification

Cette étape consiste à écrire :

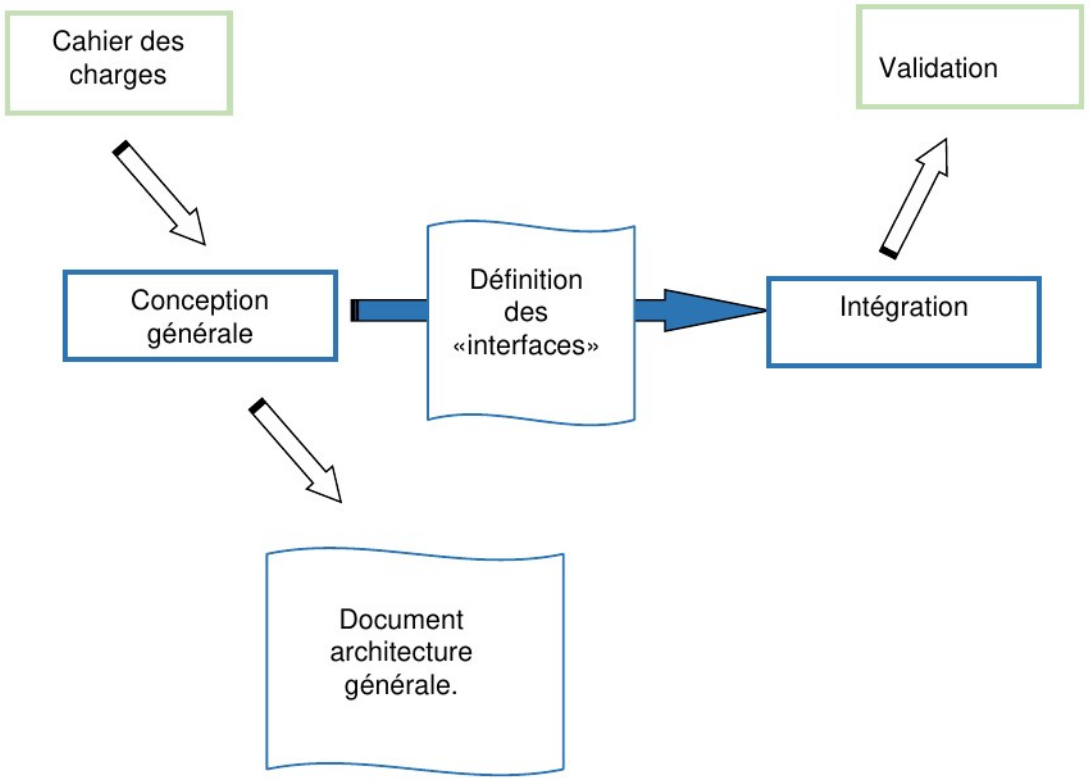
- Le cahier des charges ;
- La fiche avec les tests de validation du cahier des charges.



1.3 Niveau conception générale

Cette étape consiste à :

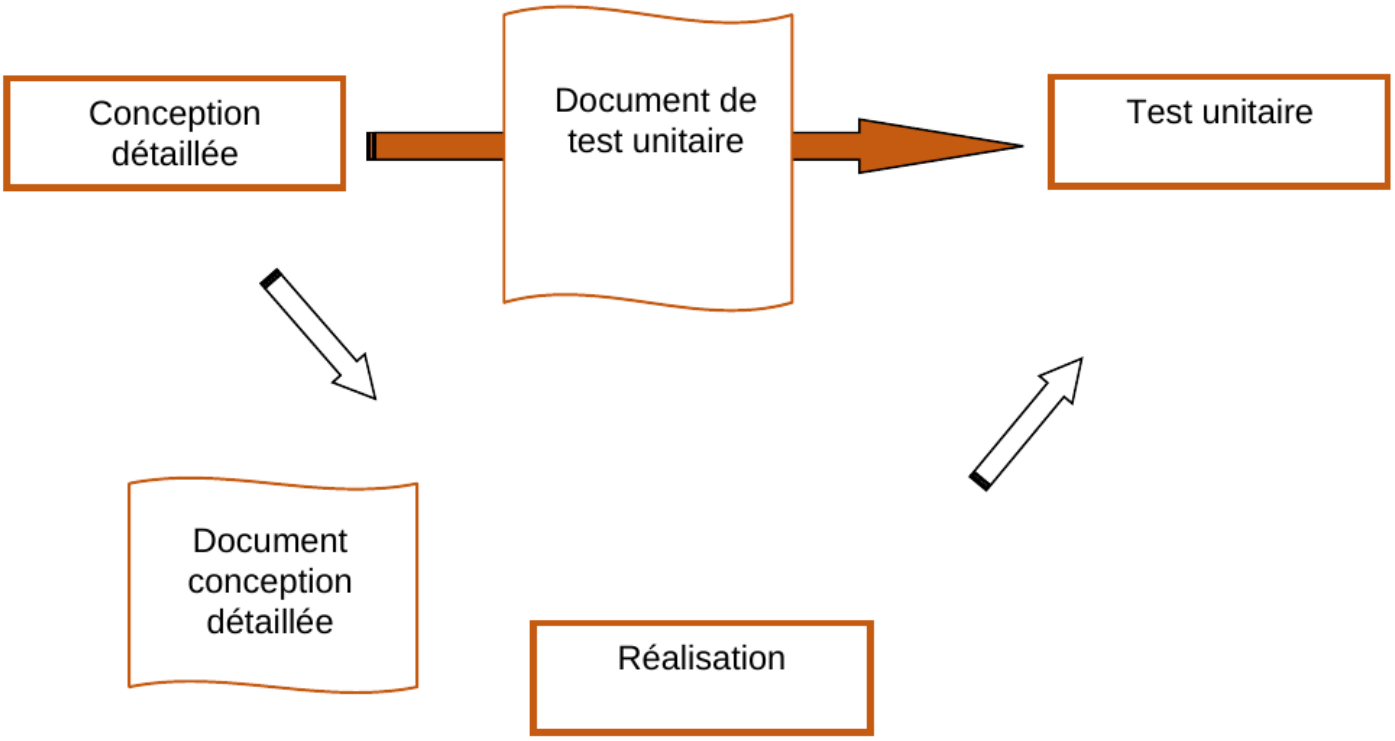
- Produire un algorithme général ;
- Définir la structure des données ;
- Définir les interfaces (signature des fonctions).



1.4 Niveau conception détaillée

Dans cette étape, pour chaque fonction :

- Proposer une structure algorithmique simple ;
- Définir les classes (en terminale).



1.5 Réalisation, codage et test unitaire

- Réaliser le codage de chaque fonction.
- Mettre au point en utilisant le débogueur.
- Ajouter des tests avec des assertions
- Vérifier les tests.

1.6 Un peu de souplesse dans la méthode

Remarque :

On peut procéder suivant une méthode dite « Agile » c'est-à-dire réaliser une intégration du projet avant le développement complet de tous ses composants. Et ainsi vérifier un fonctionnement dégradé.

On aura dans ce cas plusieurs versions du projet.

Exemple : dans le cas d'un casse brique

- Avec une seule brique.
- Avec la balle qui déplace sans détruire les briques.
- Sans compter les points.

2 Illustration à travers un exemple

2.1 Spécification

2.1.1 Expression du besoin

Analyse de la demande
Description du but ou du rôle du projet.

Exemple : jeu light_out
Le jeu light_out permet à l'utilisateur de jouer pour :

- s'initier ;
- s'amuser ;
- s'améliorer.

Analyse de l'existant
Décrire des objets, des applications qui réalisent la même fonction.
On donnera le cadre d'utilisation et les caractéristiques principales.

Exemple : jeu light_out
Le jeu existe de puis 1995. Le fonctionnement était autonome, donc sans ordinateur.

Le jeu peut se jouer sur internet :

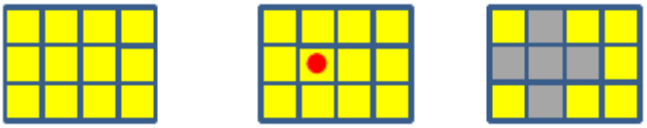
- https://www.lightsout.ir/

2.1.2 Le cahier des charges

Le cahiers des charges permet de décrire le fonctionnement et le contour de l'objet.

Exemple : jeu light_out
Le cahier des charges est ici donné sous la forme des règles du jeu light_out.

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.



Le jeu simplifié
- Les cases 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 rafraîchi.
- Phase de fin : le jeu se termine soit par une grille gagnante soit par un nombre de coups maximum.

Niveau de réalisation

Le projet light_out permet à l'utilisateur de jouer au jeu light_out sur un ordinateur. Deux versions sont envisagées :

- Une version en console (obligatoire) ;
- Une version en mode graphique (pour aller plus loin).

2.1.3 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.

2.1.4 Fiche de validation

Décrire l'ensemble des tests que doit vérifier l'objet pour fonctionner en respectant le cahier des charges.

Exemple : jeu light_out

Par un jeu de test permettant la validation du projet.
On peut proposer :

- de jouer une partie quelconque ;
- quelques parties types pour vérifier la terminaison attendue :
  - Afficher Gagné ;
  - Afficher Perdu au bout d'un certain nombre de coups.

2.2 Conception générale

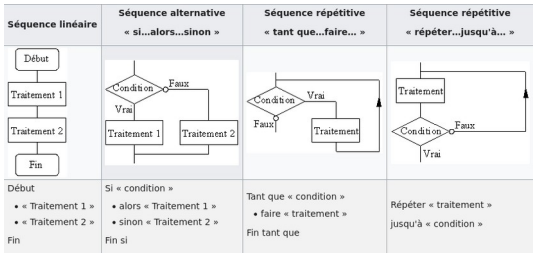
2.2.1 Consignes

- Choisir le module qui supportera l'interface graphique du jeu,
- A partir du package (module) choisi, déterminer un algorithme général,
- Proposer une organisation des données,
- Définir les objets (ne définir que les méthodes de l'objet) qui seront utilisés,
- Répartir le travail entre chaque membre de l'équipe,
- Choisir le nom des « éléments » communs (module, objet, classe, variable
- partagées, ...)

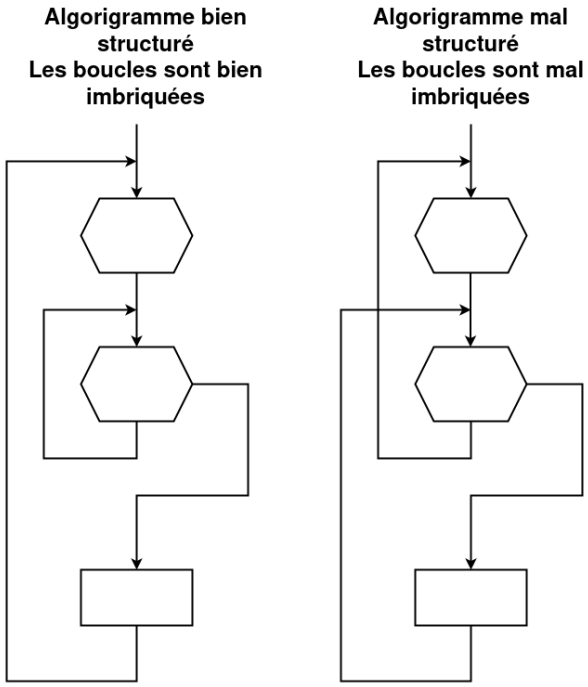
2.2.2 L'algorithme général

Dans la conception générale on peut produire :

- Une représentation graphique de l'objet (à la main ou avec un logiciel).
- Un algorigramme de fonctionnement (facultatif) :
  - Accepté pour le premier projet simple ;
  - À éviter pour les projets suivants ;
  - Peut produire des programmes non structurés ;
  - Peu lisible pour des projets complexes.
- Un algorithme général (obligatoire).
- Les flux de données entre l'utilisateur et l'objet.
- Prévoir une décomposition en modules.

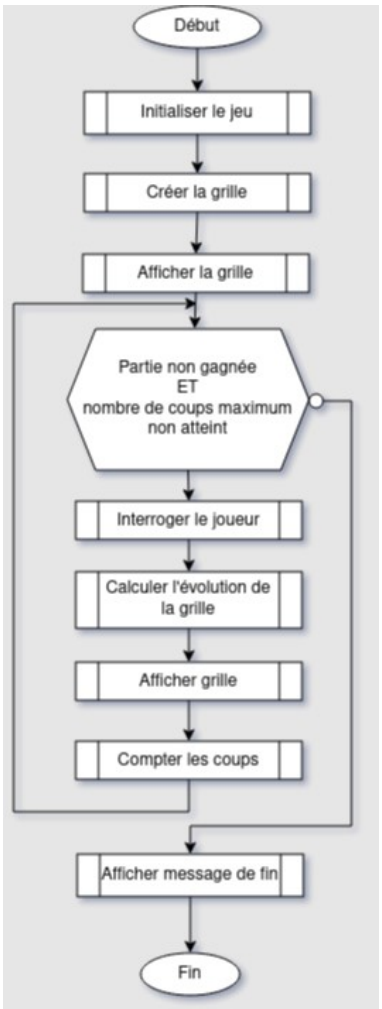


Remarque :
Les algorigrammes peuvent produire des programmes non structurés. Ce qui n'est pas le cas des algorithmes.



Exemple : jeu light_out

(facultatif) Algorigramme



(obligatoire) 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 coups 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
```

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

2.2.3 La structure des données

Avant de commencer le codage des fonctions, il faut définir la structure des données.

- La donnée qui représente l'objet ;
- Les paramètres et les renvois des fonctions ;
- Le type de donnée et son domaine de validité.

Exemple : jeu light_out

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.

```
board = [ [1, 1, 1, 1],
          [1, 1, 1, 1],
          [1, 1, 1, 1],
          [1, 1, 1, 1]]
```

Structure de donnée pour le nombre de cases limite et le maximum :

- Une constante de type entier BOX_MIN, BOX_MAX, MAX_TRIALS

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 Définition des interfaces (signatures)

- Décomposer des fonctions en sous-fonctions.
- Donner les signatures des fonctions.
  - Prototypes (paramètres d'appel, paramètres de sortie)
  - doc string
  - asserts pour les tests
- En terminale, de même pour les objets
  - Pour chaque objet et leurs méthodes définir les prototypes (paramètres d'appel) et les « doc string ». Pour certaines méthodes on définira les « asserts » pour les tests.

Exemple : jeu light_out

Le code est réparti sur 5 fichiers :

- script principal
  - light_out.py
- Modules
  - compute.py
  - game.py
  - init.py
  - screen.py

Spécifications de deux fonctions

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

    return True/False


def delta(target:int, edge:int)->tuple:
    '''
    renvoyer la variation possible de target pour trouver
    les voisins possibles en x ou y
    renvoie un tuple de la plage possible
    '''
    ...
    return answer
```

2.4 Conception détaillée

- Rechercher des méthodes algorithmiques.
- Traduire un calcul mathématique en algorithme.
- Les algorithmes doivent être simples.

2.5 Réalisation

- Traduire une méthode algorithmique en code.
- Traduire un calcul en code.
- Réaliser le codage de chaque fonction et de chaque module.
- Mettre au point en utilisant le débogueur .
- Définir le corps de : « if __name__ == __main__: »

2.6 Test unitaire

- Ajouter des tests pour vérifier les paramètres des fonctions.
- Ajouter des tests pour vérifier les renvois des fonctions.
- Vérifier que les fonctions et modules passent tous les tests.

2.7 Intégration

Réaliser l'intégration :

- En partant du programme principale.
- Ajouter les modules un par un et valider avec des tests.
- Ajouter les fonctions les unes après les autres et valider avec des tests.

2.8 *Validation*

Passer les tests définis dans la partie spécification.

Exemple : jeu light_out

On peut proposer :

- de jouer une partie quelconque ;
- quelques parties types pour vérifier la terminaison attendue :
  - Afficher Gagné ;
  - Afficher Perdu au bout d'un certain nombre de coups.

3 Travail élève et évaluation

3.1 *Travail en groupe*

- Spécification
- Conception générale
- Découpage en fonctions
- Répartition du travail et des fonctions
- Rédaction d'un document

3.2 *Évaluation du travail en groupe par le professeur*

- Évaluation du travail précédent
- Validation
- Proposition de poursuivre le projet ou de reprendre le travail de groupe

3.3 *Travail personnel*

- Chaque élève développe ses fonctions
- Teste chaque fonction
- Production :
  - Des fichiers avec le code
  - Rédaction d'un document avec l'aspect recherche et méthode :
    - Méthode algorithmique
    - Calculs théoriques

3.4 *Travail en groupe*

- Intégration
- Test de conformité
- Rédaction d'un document sur :
  - Les tests
  - La mise en conformité

3.5 *Évaluation individuelle du projet*

- Présentation du travail sous la forme d'un diaporama
- Présentation du code personnel
- Exécution du code
- Évaluation :
  - Le diaporama et son contenu
  - L'oral selon les critères du grand oral

- La conception du projet selon les critères ci-après

3.6 Critères d'évaluation

L'évaluation du projet porte sur les compétences du programme de NSI :

- 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

Lien entre la conduite de projet et les compétences.

Spécification	• Analyser et modéliser un problème
Conception générale	• Décomposer un problème en sous-problèmes, • Concevoir des solutions algorithmiques, • Développer des capacités d'abstraction et de généralisation • Mobiliser les concepts et les technologies
Conception détaillée	• Concevoir des solutions algorithmiques
Codage	• Traduire un algorithme dans un langage de programmation
Test unitaire	• Développer des capacités d'abstraction et de généralisation
Intégration	• Développer des capacités d'abstraction et de généralisation