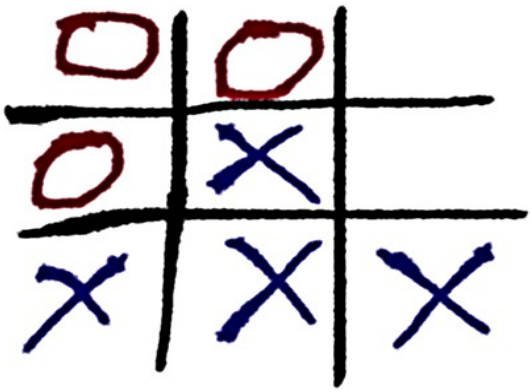
CONCEPTION



MORPION-S2

Type	**Conception**
Nom du projet	**MorpionS2**
Commentaire	**Exemple illustratif cours MDD, S2, ENIB**
Auteur	**Gireg Desmeulles**
Version	**1.0**
Date	**10/03/2014**

Table des matières

1 Rappel du cahier des charges

1.1 Contraintes techniques

- Le logiciel est associé à un cours, il doit donc fonctionner sur les machines de TP de l'ENIB pour que les élèves puissent le tester.
- Le langage utilisé en cours est Python. Le développement devra donc se faire en python.
- Les notions de programmation orientée objet n'ayant pas encore été abordées, le programme devra essentiellement s'appuyer sur le paradigme de la programmation procédurale.
- Le logiciel devra être réalisé en conformité avec les pratiques préconisées en cours de MDD : barrière d'abstraction, modularité, unicode, etc...
- L'interface sera réalisée en mode texte dans un terminal

1.2 Fonctionnalités

- F1 : Nommer le joueur
- F2 : Choisir pion
- F3 : jouer une partie
  - F3.1Jouer une manche
    - F3.1.1 Afficher le jeu :
      - grille
      - nom
      - score
      - case sélectionnée
    - F3.1.2 Se déplacer sur la grille
    - F3.1.3 Poser un pion utilisateur
    - F3.1.4 Poser un pion ordinateur
    - F3.1.5 Finir manche
  - F3.2 :Finir partie
    - F3.2.1 : Afficher résultat
    - F3.2.1 : Quitter

1.3 P1 :Prototype P1

Ce prototype porte essentiellement sur la creation de la grille et sur l'affichage.
Mise en oeuvre des fonctionnalités : F1, F2, F3.1.1, F3.1.2, F3.1.3.
Livré dans une archive au format .zip ou .tgz
Contient un manuel d'utilisation dans le fichier readme.txt

1.4 P2 :Prototype P2

Ce prototype réalise toutes les fonctionnalités.
Ajout à P1 des fonctionnalités F3.1.4, F3.1.5, F3.2.
Livré dans une archive au format .zip ou .tgz
Contient un manuel d'utilisation dans le fichier readme.txt

2 Principes des solutions techniques

2.1 Langage

Conformément aux contraintes énoncées dans le cahier des charges, le codage est réalisé avec langage python. Nous choisissons la version 2.7.5

2.2 Architecture du logiciel

Nous mettons en oeuvre le principe de la barrière d'abstraction. Chaque module correspond à un type de donnée et fournit toutes les opérations permettant de le manipuler de manière abstraite.

2.3 Interface utilisateur

L'interface utilisateur se fera via un terminal de type linux.
Nous reprenons la solution donnée en cours de MDD en utilisant les modules :
`termios, sys, select`.

2.3.1 Boucle de simulation

Le programme mettra en oeuvre une boucle de simulation qui gèrera l'affichage et les événements clavier.

2.3.2 Affichage

L'affichage se fait en communicant directement avec le terminal en envoyant des chaînes de caractères sur la sortie standard de l'application.

2.3.3 Gestion du clavier

L'entrée standard est utilisé pour détecter les actions de l'utilisateur.
Le module `tty` permet de rediriger les événements clavier sur l'entrée standard.
Pour connaître les actions de l'utilisateur il suffit de lire l'entrée standard.

2.3.4 Image ascii-art

Pour dessiner certaines parties de l'interface nous utilisons des « images ascii ».
Dans l'idée de séparer le code et les données, les différentes images ASCII seront stockées dans des fichiers textes : `victory.txt, defeat.txt, drawMatch.txt`

2.4 Grille, pions...

Pour modéliser le plateau de jeu, une liste de liste (3X3) permet de stocker des caractères correspondant au pions posés sur la grille.

Par exemple :

```
grid = [[' ','X',' '],['O','X',' '],['X','O',' ']]
#On pose un pion en 0,0
grid[0][0]='O'
```

3 Analyse

3.1 Analyse noms/verbes :

- Verbes :

  nommer, choisir, jouer, afficher, déplacer, poser, finir, quitter

- Nom :

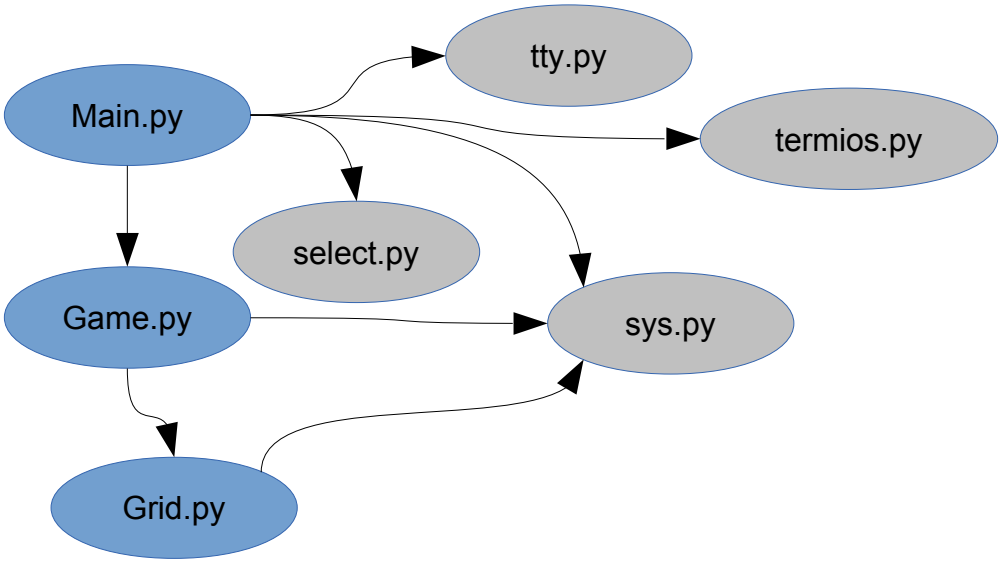
  joueur, grille, nom, score, case, pion, manche, utilisateur, ordinateur, partie, résultat

3.2 Types de donnée

```
type: Game = struct
           round      : entier
           score      : tuple (entier, entier)
           playerName : chaîne
           playerPawn : caractère
           grid       : Grid
      fstruct

type: Grid =struct
           pawns      : liste de liste caractères 3*3
           selected   : tuple (entier,entier)
      fstruct
```

3.3 Dépendance entre modules



3.4 Analyse descendante :

3.4.1 Arbre principal :

```
Main.main()
 +--  Main.init()
 |     +-- Main.askName()
 |     +-- Main.askPawn()
 |     +-- Game.create()
 |          +--Grid.create()
 |
 +-- Main.run()
       +-- Main.show()
       +-- Main.interact()
```

3.4.2 Arbre affichage

```
Main.show()
      +-- Game.show()
           +-- Grid.show()
```

3.4.3 Arbre interaction

```
Main.interact()
      |
      +-- Game.move()
      |     +-- Grid.getSelected()
      |     +-- Grid.setSelected()
      |
      +-- Game.play()
      |     +-- Grid.play()
      |     +-- Game.finishRound()
      |     |     +-- Grid.testFull()
      |     |     +-- Grid.testVictory()
      |     |     +-- Grid.create()
      |     |
      |     +-- Game.playComputer()
      |           +-- Grid.getPawns()
      |           +-- Grid.play()
      |                 +-- Grid.testFull()
      |                 +-- Grid.testVictory()
      |
      |
      +-- Game.getRound()
      +-- Main.finish()
```

4 Description des fonctions

4.1 Programme Principal : Main.py

- Main.**main()**
- Main.**init()**
- Main.**run()**
- Main.**show()**
- Main.**interact()**
- Main.**askName()**
- Main.**askPawn()**
- Main.**finish()**

Main.**main()**->rien
Description : fonction principale du jeu
Paramètres : aucun
Valeur de retour :aucune

Main.**init()**->rien
Description : initialisation du jeu
Paramètres : aucun
Valeur de retour :aucune

Main.**run()**->rien
Description : boucle de simulation
Paramètres : aucun
Valeur de retour :aucune

Main.**show()**->rien
Description : fonction principale du jeu
Paramètres : aucun
Valeur de retour :aucune

Main.**interact()**->rien
Description : gère les événements clavier
Paramètres : aucun
Valeur de retour :aucune

Main.**askName()**->chaîne
Description : demande son nom à l'utilisateur
Paramètres : aucun
Valeur de retour : le nom du joueur

Main.**askPawn()**->booléen
Description : demande à l'utilisateur de choisir son pion
Paramètres : aucun
Valeur de retour : 0 pour les « O », 1pour les « X »

Main.**finish()**->rien
Description : termine la partie, affiche le visuel de sortie en fonction du résultat
Paramètres : aucun
Valeur de retour :aucune

4.2 Game.py

- Game.**create(name,pawn)**

- Game.**getPlayerPawn(g)**
- Game.**getPlayerName(g)**
- Game.**setPlayerName(g,name)**
- Game.**getComputerPawn(g)**
- Game.**getGrid(g)**
- Game.**getScore(g)**
- Game.**getRound(g)**

- Game.**show(g)**
- Game.**play(g)**
- Game.**playComputer(g)**
- Game.**finishRound(g)**

Game.**create(name,pawn)**->Game
Description : crée une nouvelle partie
Paramètres :
name : chaîne
pawn : char
Valeur de retour : nouvelle partie

Game.**show(g)**->rien
Description: affiche le visuel de la partie en cours
Paramètres :
g : Game

Game.**play(g)**->rien
Description : pose un pion du joueur puis déclenche le jeu de l'adversaire. En cas de fin round, démarre une nouvelle manche
Paramètres :
g : Game
Valeur de retour: rien

Game.**playComputer(g)**->rien
Description : pose un pion pour l'ordinateur
Paramètres :
g : Game
Valeur de retour: rien

Game.**finishRound(g)**->booléen
Description : teste si un round une manche est terminée et démarre une nouvelle manche dans ce cas.
Paramètres :
g : Game
Valeur de retour: 0 si la manche est terminée, 1 sinon

4.3 Grid.py

- Grid.**create()**
- Grid.**setSelected(g,i)**
- Grid.**getSelected(g)**
- Grid.**getPawns(g)**

- Grid.**play(g,pawn)**
- Grid.**testFull(g)**
- Grid.**testVictory(g,pawn)**
- Grid.**show(g)**

Grid.**create()**->Grid
Description: Crée une grille vide
Paramètres: rien
Valeur de retour: grille vide

Grid.**setSelected(g,i)** → rien
Description: definit la case selectionnée
Paramètres:
g:Grid
i:tuple(entier,entier)
Valeur de retour: rien

Grid.**getSelected(g)->** tuple(entier,entier)
Description: renvoie la case sélectionnée
Paramètres:
g: Grid
Valeur de retour: index de la case sélectionnée

Grid.**play(g,pawn)**: booléen
Description: pose un pion dans la case sélectionnée
Paramètres:
g: Grid
pawn: caractère
Valeur de retour : 1 si la case était vide, 0 si impossible de jouer car la case est occupée

Grid.**testFull(g)**: booléen
Description: teste si la grille est pleine
Paramètres:
g: Grid
Valeur de retour : 1 si la case était vide, 0 si impossible de jouer car la case est occupé

Grid.**testVictory(g,pawn)**: booléen
Description: teste si trois pions sont alignés
Paramètres:
g: Grid
pawn: caractère
Valeur de retour : 1 si trois pions alignés

Grid.**show(g)**-> rien
Description: affiche la grille
Paramètres:
g: Grid
Valeur de retour : rien

5 Calendrier et suivi de développement

5.1 P1 :

5.1.1 fonctions à développer

fonctions	codées	testées	commentaires
Main.**main()**			
Main.**init()**			
Main.**run()**			
Main.**show()**			
Main.**interact()**			
Main.**askName()**			
Main.**askPawn()**			
Game.**create(name,pawn)**			
Game.**getPlayerPawn(g)**			
Game.**getPlayerName(g)**			
Game.**setPlayerName(g,name)**			
Game.**getComputerPawn(g)**			
Game.**getGrid(g)**			
Game.**getScore(g)**			
Game.**getRound(g)**			
Game.**play(g)**			
Game.**show(g)**			
Grid.**play(g,pawn)**			
Grid.**create()**			
Grid.**setSelected(g,i)**			
Grid.**getSelected(g)**			
Grid.**getPawns(g)**			
Grid.**show(g)**			

5.1.2 autre

victory.txt, defeat.txt, drawMatch.txt

5.2 P2 :

5.2.1 fonctions à développer

fonctions	codées	testées	commentaires
Main.**finish()**			
Game.**play(g)**			
Game.**playComputer(g)**			
Game.**finishRound(g)**			
Grid.**play(g,pawn)**			
Grid.**testFull(g)**			
Grid.**testVictory(g,pawn)**			