

## Avant-propos :

Le projet d' ISN que nous avons réalisé, étant conséquent, je n'irai pas forcément dans tout les détails au cours de ma présentation orale, c'est la raison pour laquelle, ce document, détaillé permettra de rappeler mes propos, mais également de les détailler.

Une seule est unique étape de la réalisation de notre projet ne sera pas détailler, il s'agit de la mise au point du design de notre boîtier, ainsi que la réalisation de ce dernier, le travail du bois, ou la réalisation de plan, ne s'inscrivant pas dans le programme d' ISN. Toutefois des documents complémentaires vous seront fournis sur ce sujet.

**Remarque :** Les crochets : **[X-y]** en gras, font référence au long de ce compte rendu, aux différent documents de l'annexe (lignes de codes, photos, schémas, ...). Le 'X' indique la partie de l'annexe(Partie **I**, Partie **II**, Partie **III**, et **Liens**) et le 'y' indique la position dans la partie de l'annexe, conformément à l'ordre d'apparitions de ces éléments dans le texte suivant. De plus, un **surlignage** indiquera au cours de ce texte des instruction Python.

## INTRODUCTION :

Au cours de cette année scolaire, nous avons été amener (Lucas WATELET, et moi-même) à réaliser un projet de fin d'année en Informatique et Sciences du Numérique pour le baccalauréat. Ce projet étant comme pour le reste de nos épreuves de l'examen, un symbole et un témoin de nos compétence acquise dans la matière, nous étions partis presque au début de l'année sur **l'idée que notre projet pourrait avoir un rapport, avec tous les thèmes vu en ISN**. Ayant déjà quelques connaissances en électronique et programmation, j'ai pu informé Lucas de l'existence d'un mini ordinateur, le Raspberry Pi, qui faisant l'interface entre l'électronique, et l'informatique nous permettait de réunir, trois thèmes étudiés au cours de l'année au sein de notre projet :

- L'architecture informatique : Le Raspberry Pi est entièrement intégré à notre projet, il s'agit par ailleurs d'un ordinateur en 'kit', laissant notre circuit imprimé visible.
- Le codage de l'information : l'interaction entre le joueur par le biais d'un boutons par exemple génère des information binaire (0 si le courant ne passe pas, 1 sinon)
- La programmation : Le Raspberry Pi fonctionne avec du Python pour interagir avec des composants exterieur (DEL, boutons, joystick)

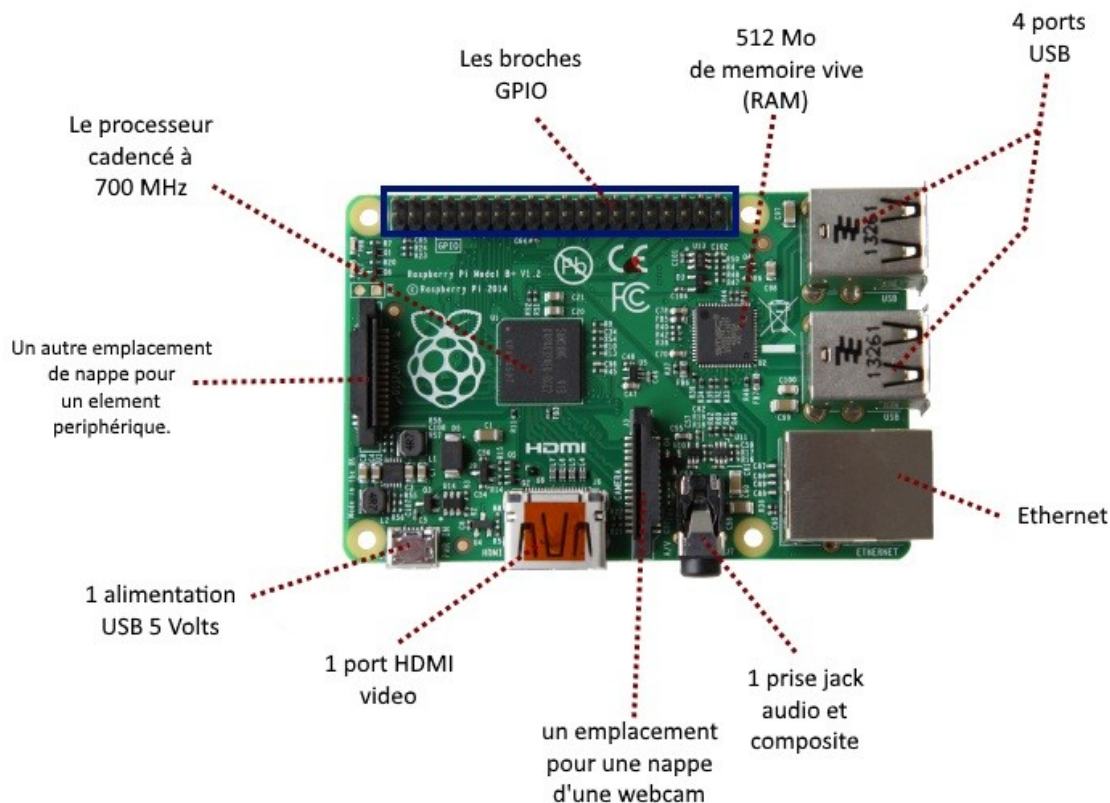
Après mûres réflexions, nous avons décidé de mettre au point une borne d'arcade sur laquelle on motterait au point un jeu de bataille navale. Afin de regrouper tout ces thèmes sans que pour autant la réalisation de ce projet devienne trop fastidieuse.

Ainsi nous diviseront notre présentation en trois partie, selon les thèmes énoncé précédemment, la dernière partie, représentant la majeure partie du travail fourni, sera plus longues que les autres et sera donc traitée en deux sous parties.

## I. L'architecture informatique au cœur de notre projet.

Notre projet est avant toutes choses réalisable grâce au 'mini' ordinateur dont je vous ai parlé précédemment. Il s'agit du Raspberry Pi B+, un ordinateur dit mono-carte, c'est-à dire que tout ces composant vitaux (RAM, processeur, circuit d'alimentation) sont soudés à la carte, et qu'il n'y existe pas de port PCI ou PCI-Express pour y joindre une carte supplémentaire (graphique, réseau,...).

De plus ces performances sont plus faibles que des ordinateurs traditionnels, en effet : son processeur n'est cadencé qu'à **700 MHz**, il ne dispose que de **512 Mo de RAM** et ne dispose pas de mémoire morte (HDD ou SSD) mais seulement d'un emplacement MicroSD.



*Ordinateur monocarte utilisé, le Raspberry Pi B+*

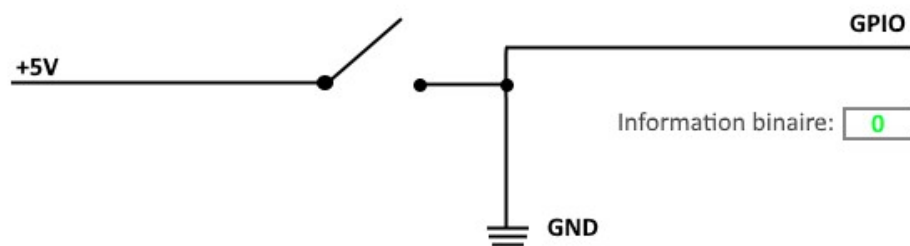
Toutefois, le Raspberry Pi même s'il peut être utilisé en ordinateur de bureau traditionnel, grâce à son système d'exploitation Linux, il est davantage utilisé dans des projets réalisés par des particuliers ou des professionnels en phase de prototypage, liant programmation et électronique (domotique, électronique embarquée, et objets connectés en général). En effet la mono-carte propose 40 'pins' (broches) GPIO [I-1], permettant la communication entre l'ordinateur et des composants électroniques via des messages binaires. À cela, s'ajoute son faible coût permettant à des amateurs comme nous de pouvoir réaliser des projets, sans prendre trop de risques.

## II. Les claviers et les souris étant trop modernes, le joystick et les boutons prennent le relais.

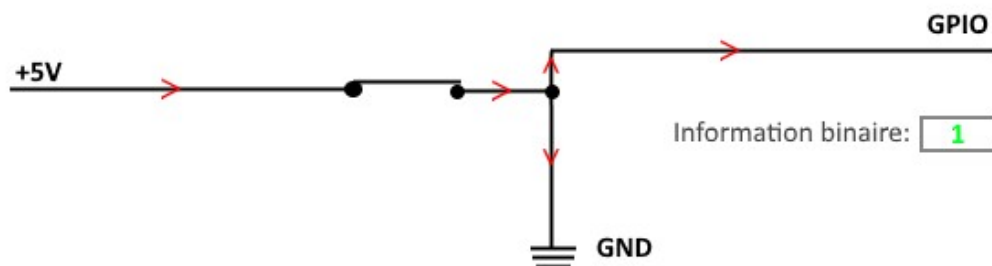
Nous avons intégré le thème de **codage de l'information** vu au cours de l'année, grâce au fait que nous avons ajouté à notre projet des boutons composants électroniques passifs, et pièces incontournables d'une borne d'arcade.

Dans cette partie je vais vulgarisé la description électronique afin d'être le plus clair possible(pour un schéma plus complet se référer à l'annexe [II-1]). Il faut donc s'imaginer un interrupteur relié en permanence à la borne positive des GPIO du raspberry Pi (+5V  $\Rightarrow$  voir annexe [I-1]) et de l'autre côté de ce dernier deux câbles, l'un relié au pôle négatif (GND en annexe) et l'autre à une entrée /sortie logique communément appelé GPIO et chargée de capté un signal positif, ou nul.

En effet si le circuit est ouvert et donc le bouton n'est pas enclenché, le signal perçu par l'ordinateur sera '0'.



Dans le cas échéant, si le bouton est pressé, le signal sera matérialisé par un '1', puisque le circuit fermé laissera passé le courant jusqu'au port GPIO.

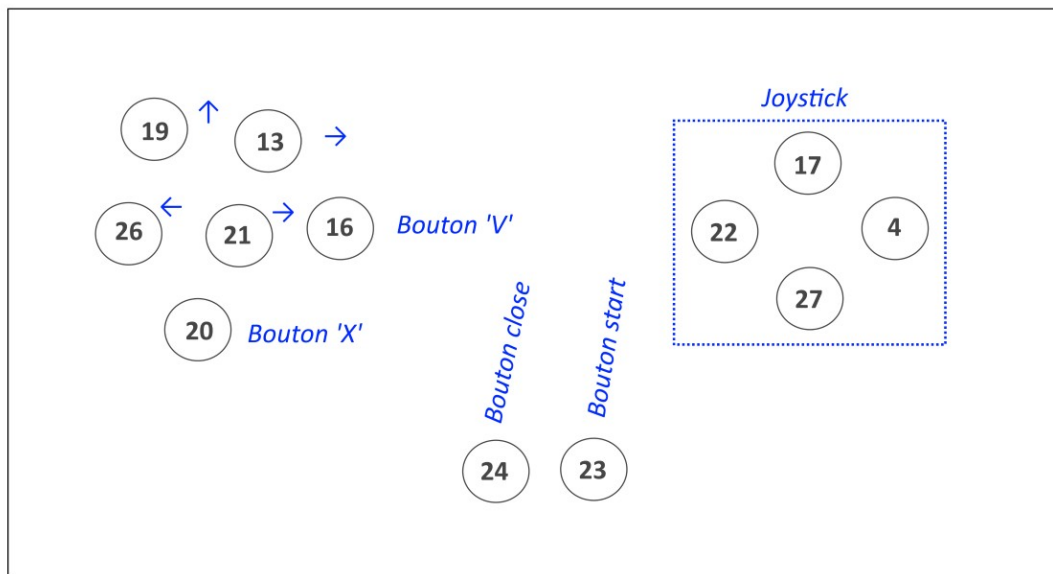


Cela illustre la transmission d'information en **langage binaire (0/1)** abordée dans le thème codage de l'information.

On attribuera ainsi pour chaque bouton un GPIO qui lui est propre selon le modèle suivant :



## Schéma du panneau de commande de la borne d'arcade



### Légende:

○ = interrupteur(bouton)    *Bouton* = Nom du bouton

42 = Numero du GPIO (BCM)

**Remarque :** on notera que le joystick correspond à un ensemble de quatre boutons, un pour chaque direction : on dira qu'il s'agit d'un joystick numérique.

### III. À l'aboutissement de notre projet, un jeu de bataille navale.

#### 1) L'idée de développement de notre jeu vidéo

##### • Les prérequis de notre idée

Il fallait que notre jeu vidéo, s'inscrive dans une thématique 'rétro', étant donnée que les bornes d'arcade sont symbolique du début du jeu vidéo grand public. C'est la raison pour laquelle on a voulu réaliser un jeu vidéo en deux dimension de **réflexion/stratégie**, la bataille navale.


Par ailleurs afin de refléter au mieux ce qui nous a été enseigné en ISN, nous avons décidé de ne pas utiliser de module de moteur graphique comme Tkinter ou Pygame afin de gérer l'aspect graphique de notre jeu nous même



avec un positionnement de chaque entité (caractère affiché par l'écran) à l'aide de liste de liste. Ces dernières, ayant été vues au cours de l'année. Ainsi, nous espérons utilisé le moins possibles de **connaissances extérieures**. Toutefois, certains ajout trouvé sur des cours de Python en ligne se sont avéré indispensables, c'est le cas pour :

- La modularité : Il nous a fallut en apprendre plus sur la modularité pour intégrer notre programme, et certaines fonctions récurrentes pour notre jeu au sein de module, pour laisser de côté, la programmation/exécution en console python, encombrante pour de long programme, et nous forçant à afficher le code lors de l'exécution de programme.  
Pour résumer le fait d'avoir programmer le jeu dans des modules nous permet qu'il s'exécute comme un logiciel traditionnel.  
**Remarque** : Le lien du cours sur la modularité suivi, est accessible en annexe, à l'adresse : [L-1].
- Le module GPIO : Le module GPIO nous permet de détecter grâce à des fonctions de ce dernier, les variation d'état de chacun des boutons de la console (Sachant que le joystick est considéré comme un ensemble de quatre boutons). Les fonctions et les instructions utilisé par sont accessible est détaillées en annexe [III-1].
- Le module OS : Le module OS va nous permettre d'importer le module src, que nous avons créer en parallèle, mais également d'introduire une fonction clef de notre programme, `os.system('clear')`, nous permettant de rafraîchir en permanence, à l'aide d'une boucle infinie notre écran.
- La module time : Le module time vient compléter la fonction précédente puisque elle permet de ralentir la fréquence d'actualisation de l'écran, pour ne pas parasité l'activité de l'utilisateur et pouvoir avoir le temps de reconnaître les signaux perçus par les GPIO avant l'actualisation de la boucle. [III-2]
- Le module sys : Ce module va nous servir pour une seule et unique fonction : `sys.exit(0)` nous permettant de fermer le programme sans l'aide d'une souris.

Par ailleurs nous avons utilisé la fonction `randint()`, du module random étudié au cours de l'année.

- Game design et développement

Pour cette partie du travail il nous a pas fallu réfléchir beaucoup, il s'agissait d'un jeu classique de **tour par tour** que l'on devra jouer contre une **IA** (Intelligence Artificielle). On devra donc penser à faire un **premier écran**, de 10x10 cases, sur lequel le joueur **placera ses bateau** (1 de six cases, 2 de quatre, 3 de trois et 4 de deux : caractères pour chaque cases d'un bateau : )

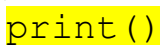
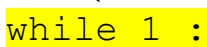
À la suite de cela l'**IA placera ses bateau** sans que le joueur ne puisse voir la position de ces derniers. Puis le **combat** pourra commencer successivement le joueur puis l'IA auront le droit à un tir sur un tableau de 10x10 (vide d'origine) qui se remplira avec les caractères  pour un tir réussi et  pour un tir raté.

## 2) Programmation en Python, l'aboutissement d'une idée

- Composition générale du programme

Le projet se battit autour de deux modules principaux conformément appelés '**src.py**' et '**bn.py**'. L'un, '**src.py**' nous permet de **créer nos fonctions de références** (souvent utilisées) que nous importeront dans notre module principal, '**bn.py**' dans lequel sera contenu le **jeu**.

- 'src.py', module contenant les fonctions de référence

Pour gérer le cote graphique de notre jeu, on s'est dit qu'il fallait faire une liste de liste qui serait imprimer (fonction ) à l'écran à chaque tour de notre boucle infini (cf :  : → cette boucle nous permet de rafraîchir l'écran en permanence pour donner une impression de continuité). Chaque élément de ces listes sont des caractères qui seront successivement imprimés à l'écran (comme peuvent l'être des pixels). Ainsi un simple espace sera imprimer comme un caractère vide, alors qu'un caractère sera imprimer. Ainsi la position de chaque élément à imprimer de la liste sera donnée grâce au repère cartésien formé par la liste de listes, pour appeler chacun de ces éléments, on écrira la fonction suivante :

```
liste_de_listes[ordonnée][abcsisse]
```

Les fonctions servant à modéliser l'affichage et la création d'un tableau sont détaillées en annexe **[III-3]** .



On pourra donc simuler un mouvement en remplaçant un caractère ayant pour coordonnée (x,y) par un espace et en le changeant ses coordonnées par (x',y'). C'est le rôle de la fonction `deplacement()` [III-4].

Il existe une dernière fonction dans ce fichier, la fonction `ecranDem()`, elle fonctionne un peu de la même façon que la les fonctions en [III-3] et permet d'afficher l'écran de démarrage du jeu (voir page de garde). Pour réaliser cette écran nous avons écrit le texte 'bataille navale' sur Paint et zoomé pour en distinguer les pixels. Nous avons ensuite écrit pour chaque ligne une chaîne de caractère dans notre liste faite de 0 et de 1, le 0 correspondant à un caractère vide, le 1 à un caractère `'█'`, pour ensuite imprimer cette liste sans oublier le saut de ligne : `'\n'` pour afficher le texte souhaité.

- 'bn.py', programme principal

Ce module se contruit en deux grandes parties : **L'initialisation**, et **la boucle infinie** :

La partie dite d'**initialisation** contient sur les deux première lignes, deux instructions précédées d'un diez : l'une obligatoire sous Linux pour indiquer l'arborescence vers le programme Python afin de pouvoir exécuté notre code, l'autre facultatif, permettant l'utilisation de l'encodage utf-8, ajoutant en plus des caractères ASCII, des caractères comme le 'é' ou le 'à' spécifique à la langue française.

Cette première partie définit également la totalité des variables utilisées dans le programme et définit deux fonctions :

- `verif()` , qui nous permettra de tester dans ce programme, lors du combat si un point de coordonnée (x,y) correspond à la position d'un élément d'un bateau.

- `ecranBN()` , qui complète la fonction de création de tableau présente dans 'src.py' afin d'afficher la numérotation de 01 à 10 et de A à J sur l' écran de création, de combat du joueur, et combat de l'IA.

```
===== Création =====  
  
X 01 02 03 04 05 06 07 08 09 10  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J
```

De plus cette partie du programme contient l'importation des différents module dont l'instruction : `import src` précédé d'une fonction issue du module os, `os.chdir()`, nous permettant de nous placé dans le répertoire du module 'src.py' pour limiter les bugs. On pourra donc appeler les différentes fonctions de notre module src grâce à l'instruction `src.fonctions()`.

Enfin on notera les quelques instructions du module GPIO [III-1] nous permettant de définir les broches entrée/sortie (GPIO) comme des entrées numériques reliées aux différents boutons. A la fin de l'initialisation on affiche l'écran de démarrage grâce à la fonction `src.ecranDem()` et on le laisse affiché pendant 5 secondes grâce à la fonctions `sleep()` du module time.

La seconde partie, **la boucle infinie** nous permet de rafraîchir l'écran pendant toute la durée du jeu en effet au début de cette boucle, on affiche le tableau utilisé grâce à la fonctions `afficher()` du module src, qui prend soin d'effacer le tableau précédent.

Remarque : Le tableau reste actualiser en permanence puisque contrairement à la fonction `input()`, les entrées GPIO fonctionnent comme des condition qui valent False lorsque le circuit est ouvert, ainsi l'exécution du programme n'est pas bloqué lorsque le joueur n'a pas d'activité.

A chaque tour de la boucle, le programme vérifie si le GPIO 24 est actif correspondant au bouton 'close' de la machine (rouge). Si c'est le cas, l'algorithme affiche un écran simple de clôture, avec une demande de confirmation, l'utilisateur peut naviguer de haut en bas en positionnant son curseur ('>>') avec le joystick (cela requiert les GPIO 27 et 17) et tant qu'il n'a pas appuyer sur le bouton valider, matérialisé par le bouton bleu (GPIO 23) le programme enregistre ses déplacements. Dès lors qu'il appuie sur le bouton pour valider, la boucle s'arrête (pas la boucle infinie, voir [III-5]), et son choix 'OUI' ou 'NON' est enregistré, une condition vérifie à la suite de cette boucle si il a valider la fermeture du programme. Si tel est le cas, la commande

`sys.exit(0)` est lancée, et le programme se ferme, sinon l'exécution de la boucle infinie continue normalement. Cette condition est accessible en annexe [III-5], de plus cette référence à l'annexe servira d'exemple pour tout le reste du code, puisqu'elle permet bien de comprendre le fonctionnement de chaque partie du programme.

La boucle infinie génère automatiquement (si le positionnement des bateau n'a pas déjà été effectuer) un écran dit de 'création', permettant à l'utilisateur de positionner ses bateau. Suite à cela la variable booléenne, autorisant le joueur a placer ses bateaux, et le joueur ne pourra plus en positionner un seul jusque à la fin du jeu. La position de ses bateaux est sauvegardé dans une liste de tuples (=liste non modifiable) (x,y) pour chaque élément de chaque bateau. Suite à cela, un écran de chargement apparaît pendant que l'IA place ses bateaux : il en place le même nombre que le joueur, et de la même taille. Cela générera comme pour le joueur une liste de tuples de position.

Puis le combat peut commencer, sur un nouvel écran vierge(généré par la fonction `ecranBN()` ) pour le joueur et l'IA.

On pourra alors déplacé le curseur `'_'` jusqu'à la position souhaité pour tirer. Dès que le bouton 'v' (blanc) est enclenché, le programme vérifie si le tir correspond à une position ennemi la position est effacé de la liste de tuples, l'écran affiche le caractère `'X'` accompagné du message 'touché'/'touché-coulé' sinon rien est effacé, le caractère `'~~'` apparaît à la position donnée et le message 'à l'eau' est affiché.

Le système fonctionne de la même façon que pour le joueur à l'exception que l'IA effectue ses tirs aléatoirement grâce à la fonction `randint()` , qu'il lui donnera une case à visé.

Le gagnant est déterminé dès que son adversaire n'a plus aucun éléments dans sa liste de tuples. La boucle infinie est alors cassé par l'instruction `'break'` et l'identité du gagnant révélé.

## IV. Conclusion : Problèmes majeurs & avenir du projet

Je traiterais dans cette partie que des problèmes majeurs que nous avons rencontré, et non des éléments de détails qui ont pu être résolu le jour même.

L'un des premier problèmes fut le rafraîchissement de l'écran pour palier à cela, je me suis rappeler de la fonction en BASIC Casio : *ClrText*, le '*Clr*' signifie 'clear' et en tapant ce mot clef sur Google avec des mot comme 'Python',... J'ai fini par trouver la fonction `os.system('clear')`, en équivalent Python sur un forum américain [L-2].

Il nous a fallu également les fonctions `break` (pour casser la boucle), `sys.exit(0)` (pour quitter le programme) et `time.sleep()` (pour marquer une pause), que nous avons trouvé contrairement à la précédente assez rapidement.

Un autre problème que j'ai put résoudre fut le problème de portée des variables en effet comme cela est expliquée sur le site d' OpenClassrooms [L-3] , une liste ne peut servir de référence à deux (ou plus) car quand l'une est modifié au cours de l'exécution du programme, les autres le seront aussi, c'est la raison pour laquelle après que j'ai trouvé le problème grâce a mes recherches, Lucas a proposé de créer une fonction permettant de créer le tableau (liste de liste) de référence pour la création de chacun des tableau du jeu, c'est la fonction `ecranBN()`.

⇒ *Un futur pour notre projet ?*

Nous avons d'ores et déjà plusieurs idée que nous n'avons pas intégrer à notre projet de fin d'année, pour ne pas que notre projet soit trop long à présenter, il s'agit par exemple d'améliorer l'intelligence de l'IA pour que dans un premier temps, il soit capable de réagir comme un joueur réel et pas avec une simple génération de tirs aléatoires, il faudrait ensuite réalisé un système de meilleurs scores basé sur le nombre de tours joué grâce à la fonction de référence `open()`.