**Sujet de projet info UE2.1**

**Année 2019-2020**

À envoyer au plus tard le **5 février 2019 à « @ : toumiab@ensta-bretagne.fr)**

**Ou à déposer à l’adresse url https://moodle.ensta-bretagne.fr/course/view.php?id=666**

**Titre**

Bataille navale en RTS multi-joueurs

Proposé par (Nom et prénom) : FROEHLICH Alexandre, LEINEN Guillaume

* Sujet libre proposé par des élèves\* : oui  non 

Si oui :

* Binôme : (Noms et prénoms) : FROEHLICH Alexandre, LEINEN Guillaume

--------------------------------------------------------------------------------- groupe : 1

* Encadrant : Abdelmalek Toumi

\*A déposer sur moodle : <https://moodle.ensta-bretagne.fr/course/view.php?id=666>

**Description du sujet (une à deux pages)**

Nous souhaitons développer un RTS (real time strategy) sur le naval avec une carte basée sur la rade de brest et des bateaux inspirés des bâtiments de la marine nationale.   
Une connexion à un serveur (codé lui aussi en python) pour gérer la partie multijoueur et le processus des entités du jeu.

Le projet se décompose alors en deux parties : le serveur qui s’occupe du calcul des déplacements, des interactions joueurs et entités ainsi que du comportement de ces dernières ; la seconde partie est le client qui s’occupe alors du rendu de la carte, de sa position et celles des entités, ainsi que des statistiques du joueur et ses actions possibles.

Le client contiendra la fenêtre de jeu, une mini-map de la zone, des statistiques sur le navire joué et une chat box. (voir schémas). Concernant le serveur, une instance tournera chez moi et les utilisateurs pourront créer une instance en local (pour éviter le problème de l'ouverture des ports). Eventuellement, les statistiques des joueurs seront retenues et stockées dans une BDD hébergée chez moi (structure MySQL tournant sur un NAS).

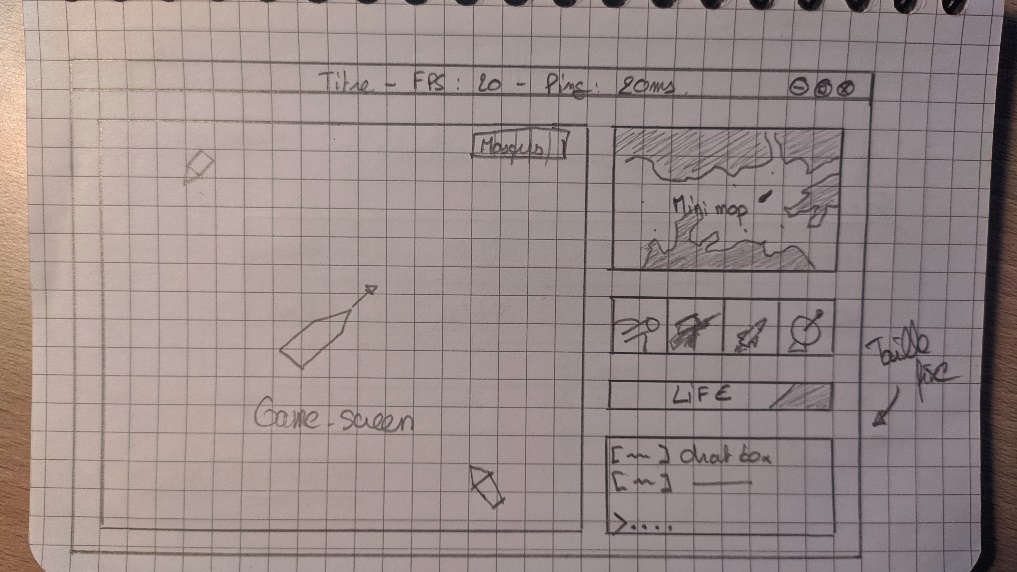
Les bibliothèques utilisées permettront le rendu de l’IHM, les structures de données et la sérialisation de ces dernières pour les envoyer par un socket TCP entre le serveur et les clients, matplotlib pour récupérer in fine les statiques sur le jeu/la partie, la connexion avec la BDD et enfin une bibliothèque pour gérer les flux d’information entrants sur le serveur.

Au niveau du fonctionnement, le client à son ouverture proposera une liste de serveurs prédéfinis ainsi que la possibilité de rentrer l’adresse IP d’un serveur en local (cette adresse s’affichera directement sur la fenêtre du serveur pour plus de confort).

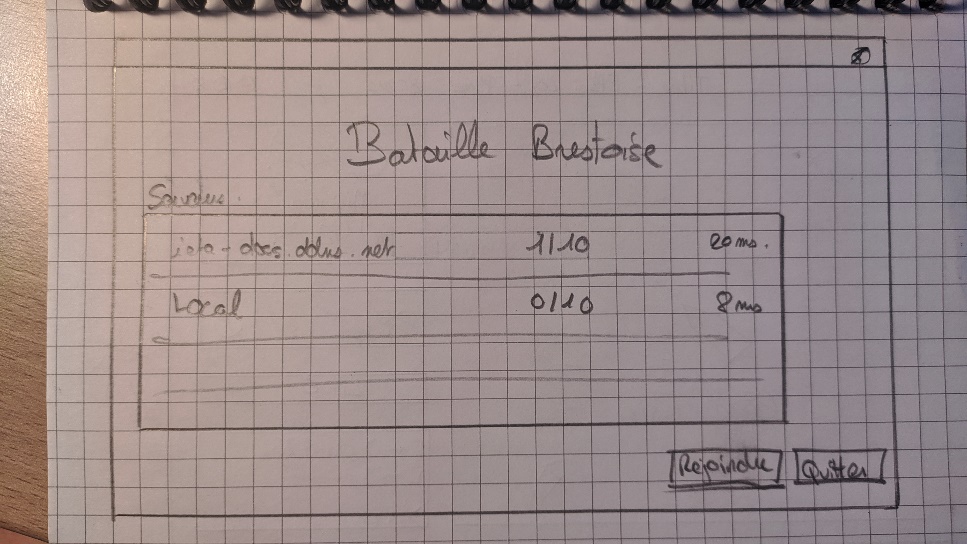
Une fois connecté au serveur, ce dernier attribut un identifiant unique au client permettant de l’identifier sur la carte. Le client peut alors choisir le bateau pour sa partie et enfin apparaitre sur la carte de jeu.

Même si dans ce prototype on ne considère pas que la triche soit importante dans ce POC (Proof Of Concept), on pourra tout de même s’arrêter sur les systèmes mis en place dans l’industrie du jeu vidéo. Pour ce projet nous avons décidé d’adopter une architecture avec un serveur authoritaire et des clients « stupides » (<https://www.gabrielgambetta.com/client-server-game-architecture.html>) Si le projet va assez loin, on pourra se concentrer sur la compensation de la latence en jeu et des interpolations d’entités (<https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking>).

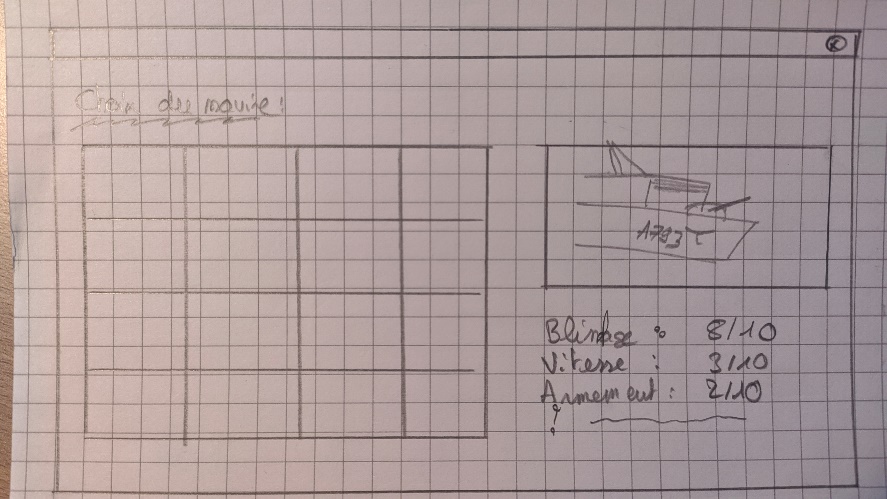
Une dernière étape qui n’est pas prévue pour le temps de ce projet, est l’implémentation d’une API et de scénarios pour une partie.



Prototype d’écran de jeu



Prototype d’écran d’accueil/de connexion



Prototype d’écran de choix de bateau

* IHM introduite dès la première partie
* **Autres modules externes** liées aux données manipulées :

Numpy, matplotlib, asyncore (ou asyncio ou la gestion socket), socket, mysql.connector, sphinx au minimum.

**Figures imposées**

* Compléter la liste des figures imposées ci-dessous (minimum 7)
* Factorisation du code : au moins trois modules et noms de classes distincts
* Documentation et commentaires du code
* Tests unitaires : (au moins 4 méthodes avec au moins 2 cas testés par méthode)
* Création d'un type d’objet (classe) : il devra contenir au moins deux variables d'instance,
* Héritage au moins entre deux types créés
* Héritage depuis un type intégré (hors en IHM)
* Fonction récursive (dans le cœur du projet)
* Structure de données dynamique (autre que celles intégrées à Python)
* Lecture/ écriture de fichiers : (type de fichier adapté – pas de « pickle »).
* Accès BDD (serveur BDD à valider avec l’encadrant)
* Utilisation de calcul vectoriel (évaluer le gain en termes de temps d’exécution).
* Autres (à préciser) :

---------------------------------------------------

----------------------------------------------------------------------------------------------

**Partie Interface Homme Machine (IHM)**

* PyQT5 (version vue en cours/td à recommander)
* tKinter
* wxPython
* PySide
* Autres : -------------------------------------------------------

**Note** : PyGame : à déconseiller

Remarque : Les codes fournis lors de la réalisation de projet seront impérativement en **Python 3.6.1**