Faculté des Sciences & Techniques — Université de Limoges

Licence 3ème année

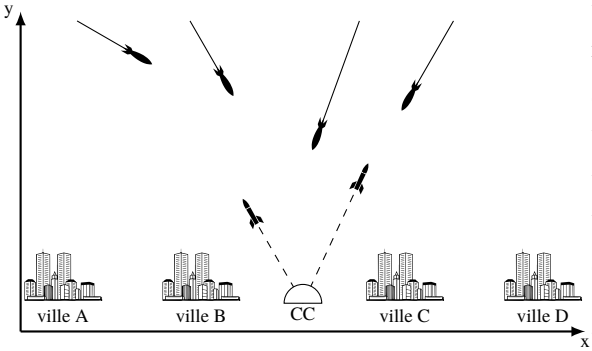
Parallélisme I

Examen — décembre 2019

Durée : 2h — Documents autorisés

Programmation asynchrone — (10 points)

1 – 10pts Vous devez programmer un jeu de « *Missile Command* » :



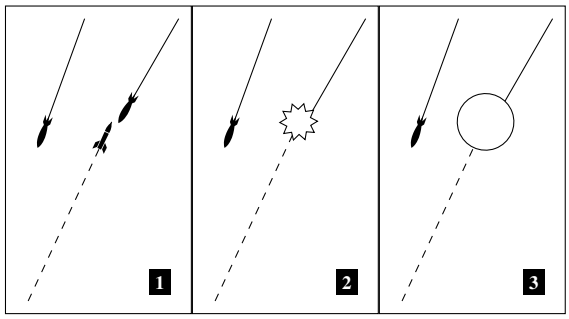
Vous avez 4 villes A, B, C et D à protéger de l'arrivée de bombes

Chaque bombe descend suivant une trajectoire **rectiligne** et une **vitesse constante**.

Vous lancez des missiles en espérant intercepter ces bombes.

Chaque missile est envoyé depuis le CC, « *Central Command* », suivant une trajectoire rectiligne et une vitesse constante.

Lorsqu'une **collision** est détectée, **1**, une explosion se produit qui détruit le missile et la bombe, **2**, puis produit une **onde de choc**, représentée par un disque qui grossit **3** :



- ▷ toute bombe qui entre en collision avec le disque de l'onde de choc explose également et produit un également un disque d'onde de choc ;
- ▷ le disque de l'onde de choc grossit jusqu'à une certaine taille avant de s'arrêter et de disparaître.

Chaque bombe et missile dispose de :

- sa position donnée par ses coordonnées (x, y) exprimées en javascript sous forme d'un tableau `[x,y]` :
  - ◇ si `c=[x,y]` alors `c[0]` correspond à x
  - ◇ et `c[1]` correspond à y

  Toutes les coordonnées x et y sont entières.
- son angle de progression qui reste constant :
  - ◇ de 10 à 170°pour un missile qui monte
  - ◇ de -10 à -170°pour une bombe qui descend ;

Une **collision** a lieu si les coordonnées d'une bombe et d'un missile sont les mêmes.

- ☐ `collision_disque(a,b,r)` qui retourne `true` s'il y a collision et `false` sinon, où :
  - ◇ `a` est le premier point donné par ses coordonnées `[x1,y1]`
  - ◇ `b` est le second point donné par ses coordonnées `[x2,y2]`
  - ◇ `r` est le rayon du disque de centre `b`
  - ◇ exemple : `collision_disque([1,2],[6,2],6)` retourne `true`.
- ☐ `progression(p,a)` qui à chaque « *tour de jeu* » permet de mettre à jour la position :
  - ◇ où p sont les coordonnées du missile ou de la bombe et `a` son angle initial ;
  - ◇ qui retourne les nouvelles coordonnées du point sous forme `[x,y]`.
- ☐ `affiche_missile(pos)`, `affiche_bombe(pos)` et `affiche_disque(pos)` qui produiront un affichage à l'écran (pour l'explosion, on utilisera l'affichage d'un disque de rayon initial 0.5).

Travail demandé : Chaque **bombe** et chaque **missile** va être **géré** par une **fonction asynchrone**.

a. Comment gérer de manière **équitable** chaque fonction asynchrone « *missile* » et « *bombe* » ? *(1pt)*

b. Comment allez vous gérer les variables des missiles et des bombes pour la mise à jour de leur **position** et la découverte de **collision** ? *(2pts)*

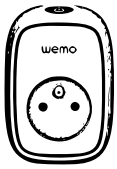
c. Lors d'une collision, le missile et la bombe ne doivent **plus être mis à jour** comment le faire ? *(1pt)*

d. Le disque d'onde de choc augmente de 0.5 à chaque tour de jeu, jusqu'à 5, comment le gérer ? *(1pt)*

e. Écrire un **programme asynchrone équitable** permettant de gérer 5 bombes et 5 missiles dont les positions et l'angle initial seront choisi de manière aléatoire, ainsi que les collisions, explosion et disque. *(5pts)*

Threads & Sémaphores – (10 points)

2 – 10pts Une société d'IoT, « *Internet of Things* », fabriquant des prises de courant connectées vous contacte pour l'aider à écrire le logiciel de son système embarqué : c-à-d un seul programme « *multi-threadé* ».



Les fonctions du système embarqué sont les suivantes :

- gestion d'un relais mécanique pour l'allumage ou l'extinction de la prise connectée ;
- un bouton physique sur la prise permettant d'allumer ou éteindre directement la prise ;
- un composant matériel WiFi :
  - fonctionnant en mode AP, « *Access Point* », pour permettre à un smartphone de se connecter et configurer la prise connectée ;
  - fonctionnant en mode STA, « *Station* », où elle se connecte au réseau WiFi de l'utilisateur qui pourra la contrôler depuis Alexa, Google Home ou son smartphone ;
- des composants logiciels :
  - un serveur Web à activer en mode AP, et STA, qui sert les « *routes* » :
    - `/` page d'accueil d'utilisation ;
    - `/config` soumettre la config. du point d'accès de l'utilisateur : nom_réseau+mot_de_passe
    - `/relais` qui déclenche le relais en extinction ou allumage ;
  - un composant « *timer* » qui permet d'attendre un certain temps avant d'activer une fonction donnée ;

L'API, « *Application Programming Interface* », est la suivante :

`activer_wifi_mode(int mode)`	AP: 0,STA: 1	active le WiFi dans le mode choisi
`cfg_wifi_sta(char *id,char *mdp)`		configure le WiFi pour se connecter
`int get_status_wifi_sta()`	0: échec, 1:succès	savoir si la connexion WiFi a fonctionné
`activer_route(char *route,int bascule)`	bascule:0 ou 1	active ou désactive la route donnée par son chemin
`activer_relais(int bascule)`	bascule:0 ou 1	allume, 1, ou éteint, 0, la prise
`timer(int duree,void *f)`	f: fonction	déclenche après un durée donnée en seconde la fonction `f`

Travail

a. Pour désactiver/activer une route dans la thread « `Web` », il faut stopper cette thread d'abord. *(3pts)*
Comment faire pour stopper la thread « `Web` », configurer la route, puis la réactiver ?
Vous donnerez le code de la fonction à utiliser et le code à ajouter à la fonction Web qui contient une boucle infinie autour de / travail web */*

b. Comment utiliser la fonction « `timer` » pour désactiver la route « `/config` » après un certain délai ? *(2pts)*

c. Le fonctionnement de la prise est réalisé par la thread « `contrôle` » : *(4pts)*
   A. premier allumage : passage en mode AP, activation des routes « `/` », « `/config` » ;
   B. attente de 600s : désactivation de la route « `config` » pour éviter une configuration non autorisée ;
   C. lors de la réception d'une requête « `/config` » le serveur Web configure le mode STA
   D. Une fois la configuration WiFi réalisée avec « `cfg_wifi_sta` », il faut utiliser la fonction « `activer_wifi_mode` » pour l'utiliser.
   Au bout de 5s, on peut utiliser la fonction « `get_status_wifi_sta` » pour savoir si cela a fonctionné.
   E. dans le cas d'un succès de la connexion :
      - on allume/éteint deux fois la prise avec la fonction « `activer_relais` » ;
      - on désactive la route « `/config` » et on active la route « `/relais` » ;
      - on a fini et on laisse fonctionner la thread « `web` » ;
   F. dans le cas d'un échec de la connexion :
      - on allume/éteint une fois la prise avec la fonction « `activer_relais` » ;
      - on recommence en A.

   Vous écrirez le code de la thread « `contrôle` », ainsi que les autres fonctions nécessaires au fonctionnement du « `timer` » et les ajouts de code à réaliser dans les threads « `wifi` » et « `web` ».
   Vous indiquerez avec des commentaires où ajouter votre code dans les threads « `wifi` » et « `web` ».

d. Comment pourrait-on gérer le bouton physique de manière efficace ? *(1pt)*