Université de Bordeaux
Licence informatique

27 avril 2018

Rapport

Projet réseau TM1A

AMEEUW Vincent
CERUTTI Marc

Résumé

Rapport pour le projet de l'enseignement
'4TIN401U - Réseaux Info L2' (2017 - 2018) sur la mise en réseau du jeu
Bomberman

Table des matières

Première partie

Preambule

Dans le cadre de l'enseignement '4TIN401U - Réseaux Info L2' (2017 - 2018) à l'Université de Bordeaux, en semestre 4 de Licence Informatique, nous avons dut adapter le jeu *Bomberman* fait grâce à la bibliothèque Pygame en multijoueur (Description en A).

Le rendu final de fin d'année fut donc d'avoir un jeu *Bomberman* fonctionnel en langage Python, avec un rapport fait sur notre travail avant le **le vendredi 27 avril à 23h55**.

Les principaux objectifs de cet enseignement était de nous familiariser sur la mise en réseau de projets informatiques. Il nous as permis ainsi de mettre en pratique nos connaissances théorique sur le réseau, la gestion des ports logicielles, des sockets, de l'envoi et de la réception de données ainsi que de son traitement.

Les contraintes techniques étaient de le faire à l'aide d'un serveur centralisé, qui ne réalise pas d'affichage graphique, mais maintient à jour l'état courant du jeu. Seuls les clients sont en charge de l'interaction avec l'utilisateur (clavier et affichage graphique)et chaque client dispose d'une copie du modèle, qu'il doit maintenir à jour à travers des échanges réseaux avec le serveur.
En d'autres termes :

— Récupération par le client du modèle serveur à travers le réseau (map, fruits, players).
— Gestion des connexions / déconnexions des joueurs.
— Gestion des déplacements des joueurs.
— Gestion des bombes.
— Extension à de multiples joueurs.
— Gestion des erreurs (mort violente d'un client, coupure réseau).
— Ajout de bonus FUN dans le jeu, impliquant de faire du réseau.

Deuxième partie

Projet réseau

1 Méthode de travail

Pour notre méthode de travail, on s'était mis d'abord d'accord sur les protocoles réseau à utiliser et le squelette du code sur papier, puis on a travailler chacun de son coté en adaptant le code de l'autre.

Notre base de code était ainsi assez modulaire pour que l'on ai pas de de problèmes sur d'éventuelles modifications ou imprévu du code pour la suite.

2 Analyse du modèle

/**/

3 Algorithme et implémentation

3.1 Protocoles

3.2 Choix techniques

4 Améliorations effectues

4.1 Collisions sur les bombes

4.2 Gestion de déconnexion

5 Bilan et critique

Troisième partie
Annexes

A Moodle

https ://moodle1.u-bordeaux.fr/course/view.php ?id=3671
https ://github.com/orel33/bomber

B Code Source

B.1 Network.py

```
# -*- coding: Utf-8 -*
# Author: aurelien.esnard@u-bordeaux.fr

import socket
import select
import threading
import sys
from model import *

################################################################################
#                              AUXILLARY FUNCTION NETWORK
                  #
################################################################################

#Size taken to the socket's buffer
SIZE_BUFFER_NETWORK = 2056
TIMEOUT = 20


class Command_Network:

    def __init__(self, model, isServer):
        self.model = model;
        self.isServer = isServer;


    '''
        #Commands
        __________

        #End for big transmissions with loops.
        END

        #Send a message to the client
        MSG <msg>

        #Send error and close the client
        ERROR <msg>

        #Connection player
        CON <nicknamePlayer>

        #Transmit map
        MAP <namemap>

        #Move player
        MOVE <nicknamePlayer> <direction>

        #Add player
        A_PLAY <nicknamePlayer> <isplayer> <kind> <posX> <posY>
    <health>

        #Add bomb
        A_BOMB <pos X> <pos Y> <range> <countdown>

        #Drop Bomb
        DP_BOMB <nicknamePlayer> <range> <countdown>

```

```python
        #Add fruit
        A_FRUIT <kind> <pos X> <pos Y>

        #Synchronisation of life
        S_LIFE <nicknamePlayer> <health>

        #Kill player
        KILL <nicknamePlayer>

        #Disconnection of the client
        QUIT <nicknamePlayer>

        #TOADD
        -send map


    '''
    '''
    Encode les commandes pour l'envoi réseau.
    En cas de commande inconnu, retourne None.
    '''
    def enc_command(self, cmd):
        cmd.replace('\\','')

        #print ("ENC")
        #print (cmd)
        #print ()

        if cmd.startswith("CON"):
            cmd = cmd.split(" ")
            return str("CON " + cmd[1] + " \\").encode()

        elif cmd.startswith("MSG"):
            cmd = cmd.partition(" ")
            return str("MSG " + cmd[2] + " \\").encode()

        elif cmd.startswith("ERROR"):
            cmd = cmd.partition(" ")
            return str("ERROR " + cmd[2] + " \\").encode()

        elif cmd.startswith("MAP"):
            cmd =cmd.split(" ")
            return str("MAP " + cmd[1]  +" \\").encode()

        elif cmd.startswith("A_PLAY"):
            cmd =cmd.split(" ")
            return str("A_PLAY " + cmd[1] + ' ' + cmd[2] + ' ' + cmd[3] +  ' ' + cmd[4] + ' ' + cmd[5]+ ' ' + cmd[6] + " \\").encode()

        elif cmd.startswith("MOVE"):
            cmd = cmd.split(' ')
            return str("MOVE " + cmd[1] + ' ' + cmd[2] + " \\").encode()

        elif cmd.startswith("A_BOMB"):
            cmd =cmd.split(" ")
            return str("A_BOMB " + cmd[1] + ' ' + cmd[2]  + ' ' + cmd[3]+ " \\").encode()

        elif cmd.startswith("DP_BOMB"):
            cmd =cmd.split(" ")
            return str("DP_BOMB␣" + cmd[1] + '␣' + cmd[2]  + '␣' + cmd[3]+ "␣\\").encode()

        elif cmd.startswith("A_FRUIT"):
            cmd =cmd.split("␣")
            return str("A_FRUIT␣" + cmd[1] + '␣' + cmd[2]  + '␣' + cmd[3]  +"␣\\").encode()

        elif cmd.startswith("S_LIFE"):
            cmd = cmd.split('␣')
            return str("S_LIFE␣" + cmd[1] + '␣' + cmd[2] + "␣\\").encode()

        elif cmd.startswith("KILL"):
            cmd = cmd.split('␣')
            return str("KILL␣" + cmd[1] + "␣\\").encode()

        elif cmd.startswith("QUIT"):
            cmd = cmd.split('␣')
            return str("QUIT␣" + cmd[1] + "␣\\").encode()

        elif cmd.startswith("END"):
            cmd =cmd.split("␣")
            return str("END␣" + "\\").encode()

        return None;

    '''
    Decode les commandes.
    Adapte le modèle et renvoi une liste de string correspondant aux commandes.
    Return None en cas de commandes inconnus.
    '''
    def dec_command(self, msg):

        listCmds = msg.decode()
        listCmds = listCmds.split('\\')
        #print ("BUFFER")
        #print (listCmds)

        listValid =[]

        while (listCmds != [] and listCmds[0] != ''):

            cmd = listCmds[0]
            cmd = cmd.replace ('\\','␣')
            #print ("DEC")
            #print (cmd)
            #print ()
            del listCmds[0]

            if cmd.startswith("CON␣"):
                cmdtmp = cmd.split('␣')
                listValid.append(cmd)

            elif cmd.startswith("MSG␣"):
                cmdtmp = cmd.partition('␣')
                print (cmdtmp[2])
                listValid.append(cmd)

            elif cmd.startswith("ERROR␣"):
                cmdtmp = cmd.partition('␣')
```

```python
            print ("ERROR : "+ cmdtmp[2])
            sys.exit(1)

        elif cmd.startswith("MAP "):
            cmdtmp = cmd.split(' ')
            self.model.load_map(cmdtmp[1])
            listValid.append(cmd)

        elif cmd.startswith("MOVE "):
            cmdtmp = cmd.split(' ')
            nickname = cmdtmp[1]
            direction = int(cmdtmp[2])
            if direction in DIRECTIONS:
                try:
                    self.model.move_character(nickname, direction)
                except:
                    listValid.append(str("MSG You are dead !!"))
                    pass
            listValid.append(cmd)

        elif cmd.startswith("A_PLAY "):
            cmdtmp = cmd.split(' ')
            self.model.add_character(cmdtmp[1],bool(int(cmdtmp[2])),int(cmdtmp[3]),(int(cmdtmp[4]), int(cmdtmp[5])), int(cmdtmp[6]))
            listValid.append(cmd)

        elif cmd.startswith("A_BOMB "):
            cmdtmp = cmd.split(' ')
            self.model.bombs.append(Bomb(self.model.map, (int(cmdtmp[1]),int(cmdtmp[2])),int(cmdtmp[3]),int(cmdtmp[4])))
            listValid.append(cmd)

        elif cmd.startswith("DP_BOMB "):
            cmdtmp = cmd.split(' ')
            try:
                self.model.drop_bomb(cmdtmp[1], int(cmdtmp[2]), int(cmdtmp[3]))
            except:
                listValid.append(str("MSG You are dead !!"))
                pass
            listValid.append(cmd)

        elif cmd.startswith("A_FRUIT "):
            cmdtmp = cmd.split(' ')
            self.model.add_fruit(int(cmdtmp[1]), (int(cmdtmp[2]), int(cmdtmp[3])))
            listValid.append(cmd)

        elif cmd.startswith("S_LIFE "):
            cmdtmp = cmd.split(' ')
            player = self.model.look(cmdtmp[1])
            if player != None :
                player.health = int(cmdtmp[2])
            else:
                listValid.append(str("KILL "+cmdtmp[1]))
                pass

        elif cmd.startswith("KILL ") or cmd.startswith("QUIT "):
                cmdtmp = cmd.split(' ')
                try:
                    self.model.kill_character(cmdtmp[1]);
                    print (cmd)
                except:
                    pass

                listValid.append(cmd)

            elif cmd.startswith("END"):
                cmdtmp = cmd.split(' ')
                listValid.append(cmd)

            else:
                return None

        return listValid;







################################################################################
#                              NETWORK SERVER CONTROLLER                       #
################################################################################

class NetworkServerController:

    def __init__(self, model, port):
        self.port = port;
        self.cmd = Command_Network(model,True)
        self.soc = socket.socket(socket.AF_INET6, socket.SOCK_STREAM);
        self.soc.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1);
        self.soc.bind(('', port));
        self.soc.listen(1);
        self.socks = {};
        self.afk={}
        self.socks[self.soc] = "SERVER";

    '''
    Connection d'un nouveau client, initialise ses champs
    '''
    def clientConnection(self, sockserv):
        newSock, addr= sockserv.accept()
        msg = newSock.recv(SIZE_BUFFER_NETWORK)

        listcmd = self.cmd.dec_command(msg)

        if (listcmd!=None and listcmd[0].startswith("CON")):
            nick= listcmd[0].split(" ")[1]
            validNick = True
            Afk = False
            for s in self.socks:
                if self.socks[s]== nick and s not in self.afk:
                    print ("Error command init new player, name already use.")

    newSock.sendall(self.cmd.enc_command(str("ERROR␣command␣init␣new␣player,␣name␣already␣use.")))
                    validNick = False
                    newSock.close();
                if s in self.afk:
                    Afk=True

            if validNick :
                self.socks[newSock]= nick
                if not Afk :
                    self.cmd.model.add_character(nick, False)
                else:
                    for s in self.afk:
                        if self.socks[s]==nick:
                            self.afk.pop(s)
                            self.socks.pop(s)
                            s.close()
                            break

                print("New␣connection")
                print(addr)

                # envoyer map, fruits, joueurs, bombes
                self.initMap(newSock);
                self.initFruits(newSock)
                self.initBombs(newSock)
                self.initCharacters(newSock,Afk)
                newSock.sendall(self.cmd.enc_command(str("END␣")))
        else:
            print ("Error␣command␣init␣new␣player")
            newSock.close();

    '''
    Doit renvoyer aux autres destinataires
    '''
    def re_send(self,sockSender, cmd):
        for sock in self.socks:
            if sock != self.soc and sock != sockSender:
                try :
                    sock.sendall(self.cmd.enc_command(cmd))
                except:
                    print (self.socks[sock])
                    print (cmd)
                    print("Error␣message␣not␣have␣been␣sent.")

    '''
    Initialise les characters à envoyer
    '''
    def initCharacters(self, s, afk):
        for char in self.cmd.model.characters:
            if (char.nickname == self.socks[s]):
                #is_player = true, send for initialization to others = false
                s.sendall(self.cmd.enc_command(str("A_PLAY␣"+char.nickname+"␣"+"1"+"␣"+str(char.kind)+"␣"+str(char.pos[X])+"␣"+ str(char.pos[Y])+"␣"+ str(char.health))))
                if not afk:
                    self.re_send(s, str("A_PLAY␣"+char.nickname+"␣"+"0"+"␣"+str(char.kind)+"␣"+ str(char.pos[X])+"␣"+str(char.pos[Y])+"␣"+ str(char.health)))
            else:
                s.sendall(self.cmd.enc_command(str("A_PLAY␣
"+char.nickname+" "+"0"+" "+str(char.kind)+" "+
str(char.pos[X])+" "+ str(char.pos[Y])+" "+ str(char.health))))

    '''
    Initialise les fruits à envoyer
    '''
    def initFruits(self, s):
        for fruit in self.cmd.model.fruits:
            s.sendall(self.cmd.enc_command(str("A_FRUIT 
    "+str(FRUITS[fruit.kind])+" "+ str(fruit.pos[X])+" "+
    str(fruit.pos[Y]))))
        return
    '''
    Initialise les bombs à envoyer
    '''
    def initBombs(self, s):
        for bomb in self.cmd.model.bombs:
            s.sendall(self.cmd.enc_command(str("A_BOMB 
    "+str(bomb.pos[X])+" "+str(bomb.pos[Y])+" 
    "+str(bomb.max_range)+" "+str(bomb.countdown))))
        return

    '''
    Initialise la map à envoyer
    '''
    def initMap(self, s):
        if len(sys.argv) == 3:
            s.sendall(self.cmd.enc_command(str("MAP 
    "+sys.argv[2])));
        else:
            s.sendall(self.cmd.enc_command(str("MAP 
    "+DEFAULT_MAP)));
        return

    '''
    Déconnecte un client et renvoie le nom du joueur à supprimer
    '''
    def disconnectClient(self, s):
        if s in self.socks:
            nick = self.socks[s]
            self.cmd.model.quit(nick);
            s.close()
            self.socks.pop(s)
            self.re_send(s, str("KILL "+ nick))


    # time event

    def tick(self, dt):
        sel = select.select(self.socks, [], [], 0);
        if sel[0]:
            for s in sel[0]:
                if s is self.soc:
                    self.clientConnection(s);

                elif s in self.socks:
                    if s not in self.afk:
                        msg =b""
                        try:
                            msg = s.recv(SIZE_BUFFER_NETWORK);
                        except:
                            print ("Error interuption")
```

```python
                            print("Connection client afk.")
                            self.afk[s]=(TIMEOUT+1)*1000-1
                            #self.disconnectClient(s)
                            break

                        if (len(msg) <= 0):
                            print ("Error message empty.")
                            self.afk[s]=(TIMEOUT+1)*1000-1
                            #self.disconnectClient(s)
                            break

                        else:
                            listCmd = self.cmd.dec_command(msg)
                            for cmd in listCmd:
                                if cmd.startswith("QUIT"):
                                     self.disconnectClient(s)
                                     break
                                else:
                                    self.re_send(s, cmd)

                        for char in self.cmd.model.characters:
                             self.re_send(s ,str("S_LIFE "+str(char.nickname)+" "+str(char.health)));
                    else:
                        try:
                            msg = s.recv(SIZE_BUFFER_NETWORK);
                            self.afk.pop(s)

                        except:
                            self.afk[s]-=dt
                            print(int(self.afk[s] / 1000))
                            if (self.afk[s]<0):
                                print ("timeout connection")
                                print (self.socks[s])
                                self.afk.pop(s)
                                self.disconnectClient(s)


        return True

################################################################################
#                          NETWORK CLIENT CONTROLLER                           #
################################################################################

class NetworkClientController:

    def __init__(self, model, host, port, nickname):
        self.host = host;
        self.port = port;
        self.cmd = Command_Network(model,False)
        self.nickname = nickname;
        self.soc = None;
        try:
            request = socket.getaddrinfo(self.host, self.port, 0, socket.SOCK_STREAM);
        except:
            print("Error : can't connect to server.\n");
            sys.exit(1);
        for res in request:
            try:
                self.soc = socket.socket(res[0], res[1]);
            except:
                self.soc = None;
                continue;
            try:
                self.soc.connect(res[4]);
            except:
                self.soc.close();
                self.soc = None;
                continue;
            print("Connected.\n");
            break;
        if self.soc is None:
            print("Error␣:␣can't␣open␣connection.\n");
            sys.exit(1);

        print ("Connection␣to␣server␣open.")
        print ("Send␣request␣game␣...")
        print()
        #Connection
        self.soc.sendall(self.cmd.enc_command(str("CON␣"+nickname)));


        #Decode map + objects (fruits , bombs) + players
        stop = False
        while (not stop):

            msg = self.soc.recv(SIZE_BUFFER_NETWORK)
            if len(msg )<= 0 :
                print ("Brutal␣interruption␣of␣the␣connection␣during␣the␣chargement␣of␣the␣map.")
                sys.exit(1)

            listCmd = self.cmd.dec_command(msg)

            if (listCmd==None):
                stop = True
                print ("Unknow␣command␣give␣by␣the␣server,␣maybe␣it␣have␣not␣the␣same␣version.")
                sys.exit(1)

            for c in listCmd:
                if c.startswith("END"):
                    stop = True
                    break



    # keyboard events

    def keyboard_quit(self):
        print("=>␣event␣\"quit\"")
        if not self.cmd.model.player: return False
        self.soc.sendall(self.cmd.enc_command(str("QUIT␣"+self.cmd.model.player.nickname)))
        sys.exit()
        return False

    def keyboard_move_character(self, direction):
        print("=>␣event␣\"keyboard␣move␣direction\"␣{}".format(DIRECTIONS_STR[direction]))

```

```python
        if not self.cmd.model.player: return True

        self.soc.sendall(self.cmd.enc_command(str("MOVE "+self.cmd.model.player.nickname+" "+str(direction))));

        #SOLO
        nickname = self.cmd.model.player.nickname
        if direction in DIRECTIONS:
            self.cmd.model.move_character(nickname, direction)

        return True

    def keyboard_drop_bomb(self):
        print("=> event \"keyboard drop bomb\"")

        if not self.cmd.model.player: return True

        self.soc.sendall(self.cmd.enc_command(str("DP_BOMB "+self.cmd.model.player.nickname+" "+str(MAX_RANGE)+" "+str(COUNTDOWN))));

        #SOLO
        nickname = self.cmd.model.player.nickname
        self.cmd.model.drop_bomb(nickname)

        return True

    # time event

    def tick(self, dt):
        sel = select.select([self.soc], [], [], 0);
        if sel[0]:
            for s in sel[0]:
                try:
                    msg = s.recv(SIZE_BUFFER_NETWORK);
                except:
                    print ("Error: Server has been disconnected")
                    s.close();
                    sys.exit(1)

                if (len(msg) <= 0):
                    print ("Error: message empty, server has been disconnected")
                    s.close();
                    sys.exit(1)

                listCmd = self.cmd.dec_command(msg)
                if (listCmd==None):
                    print ("Unknow command give by the server, maybe it have not the same version.")
                    sys.exit(1)

        if self.cmd.model.player != None :
            self.soc.sendall(self.cmd.enc_command(str("S_LIFE "+str(self.cmd.model.player.nickname)+" "+str(self.cmd.model.player.health))));


        return True
```