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

Préambule

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 dû adapter le jeu *Bomberman* fait grâce à la bibliothèque Pygame en multi-joueur (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**.

Le principal objectif de cet enseignement était de nous familiariser sur la mise en réseau de projets informatiques. Il nous a ainsi permis de mettre en pratique nos connaissances théoriques sur le réseau, la gestion des ports logiciels, des sockets, de l'envoi et de la réception de données ainsi que de leur 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 au 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'est d'abord mis d'accord sur les protocoles réseau à utiliser et le squelette du code sur papier, puis on a travaillé chacun de notre côté en adaptant le code de l'autre.

Notre base de code était ainsi assez modulaire pour ne pas avoir de problèmes sur d'éventuelles modifications ou imprévus du code pour la suite.

2 Analyse du modèle

/**/

3 Algorithme et implémentation

3.1 Protocoles

3.2 Choix techniques

4 Améliorations effectuées

4.1 Collisions sur les bombes

L'un des principaux problèmes que nous avons rencontré en jouant est que les parties sont longues (il est difficile d'éliminer les autres). L'ajout de collisions avec les bombes permet de bloquer les joueurs adverses, les rendant plus simples à éliminer. Les parties sont de fait plus courtes mais avec plus d'action.

4.2 Gestion des déconnexions

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 for deconnection afk
TIMEOUT = 20


class CommandNetwork:

    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>
```

```
57
58          #Add fruit
59          A_FRUIT <kind> <pos X> <pos Y>
60
61          #Synchronisation of life
62          S_LIFE <nicknamePlayer> <health>
63
64          #Kill player
65          KILL <nicknamePlayer>
66
67          #Disconnection of the client
68          QUIT <nicknamePlayer>
69
70          #TOADD
71          -send map
72
73
74      '''
75      '''
76      Encode les commandes pour l'envoi réseau.
77      En cas de commande inconnu, retourne None.
78      '''
79      def enc_command(self, cmd):
80          cmd.replace('\\','')
81
82          #print ("ENC")
83          #print (cmd)
84          #print ()
85
86          if cmd.startswith("CON"):
87              cmd = cmd.split("␣")
88              return str("CON␣" + cmd[1] + "␣\\").encode()
89
90          elif cmd.startswith("MSG"):
91              cmd = cmd.partition("␣")
92              return str("MSG␣" + cmd[2] + "␣\\").encode()
93
94          elif cmd.startswith("ERROR"):
95              cmd = cmd.partition("␣")
96              return str("ERROR␣" + cmd[2] + "␣\\").encode()
97
98          elif cmd.startswith("MAP"):
99              cmd =cmd.split("␣")
100             return str("MAP␣" + cmd[1]  +"␣\\").encode()
101
102         elif cmd.startswith("A_PLAY"):
103             cmd =cmd.split("␣")
104             return str("A_PLAY␣" + cmd[1] + '␣' + cmd[2] + '␣' +
    cmd[3] +  '␣' + cmd[4] + '␣' + cmd[5]+ '␣' + cmd[6] + "␣
    \\").encode()
105
106         elif cmd.startswith("MOVE"):
107             cmd = cmd.split('␣')
108             return str("MOVE␣" + cmd[1] + '␣' + cmd[2] + "␣
    \\").encode()
109
110         elif cmd.startswith("A_BOMB"):
111             cmd =cmd.split("␣")
112             return str("A_BOMB␣" + cmd[1] + '␣' + cmd[2]  + '␣' +
    cmd[3]+ "␣\\").encode()
113
114         elif cmd.startswith("DP_BOMB"):
```

```
115             cmd =cmd.split("␣")
116             return str("DP_BOMB␣" + cmd[1] + '␣' + cmd[2]  + '␣' +
    cmd[3]+ "␣\\").encode()
117
118         elif cmd.startswith("A_FRUIT"):
119             cmd =cmd.split("␣")
120             return str("A_FRUIT␣" + cmd[1] + '␣' + cmd[2]  + '␣' +
    cmd[3]  +"␣\\").encode()
121
122         elif cmd.startswith("S_LIFE"):
123             cmd = cmd.split('␣')
124             return str("S_LIFE␣" + cmd[1] + '␣' + cmd[2] + "␣
    \\").encode()
125
126         elif cmd.startswith("KILL"):
127             cmd = cmd.split('␣')
128             return str("KILL␣" + cmd[1] + "␣\\").encode()
129
130         elif cmd.startswith("QUIT"):
131             cmd = cmd.split('␣')
132             return str("QUIT␣" + cmd[1] + "␣\\").encode()
133
134         elif cmd.startswith("END"):
135             cmd =cmd.split("␣")
136             return str("END␣" + "\\").encode()
137
138         return None;
139
140     '''
141     Decode les commandes.
142     Adapte le modèle et renvoi une liste de string correspondant
    aux commandes.
143     Return None en cas de commandes inconnus.
144     '''
145     def dec_command(self, msg):
146
147         listCmds = msg.decode()
148         listCmds = listCmds.split('\\')
149         #print ("BUFFER")
150         #print (listCmds)
151
152         listValid =[]
153
154         while (listCmds != [] and listCmds[0] != ''):
155
156             cmd = listCmds[0]
157             cmd = cmd.replace ('\\','␣')
158             #print ("DEC")
159             #print (cmd)
160             #print ()
161             del listCmds[0]
162
163             if cmd.startswith("CON␣"):
164                 cmdtmp = cmd.split('␣')
165                 listValid.append(cmd)
166
167             elif cmd.startswith("MSG␣"):
168                 cmdtmp = cmd.partition('␣')
169                 print (cmdtmp[2])
170                 listValid.append(cmd)
171
172             elif cmd.startswith("ERROR␣"):
```

```
173                 cmdtmp = cmd.partition('␣')
174                 print ("ERROR␣:␣"+ cmdtmp[2])
175                 sys.exit(1)
176 
177             elif cmd.startswith("MAP␣"):
178                 cmdtmp = cmd.split('␣')
179                 self.model.load_map(cmdtmp[1])
180                 listValid.append(cmd)
181 
182             elif cmd.startswith("MOVE␣"):
183                 cmdtmp = cmd.split('␣')
184                 nickname = cmdtmp[1]
185                 direction = int(cmdtmp[2])
186                 if direction in DIRECTIONS:
187                     try:
188                         self.model.move_character(nickname,
    direction)
189                     except:
190                         listValid.append(str("MSG␣You␣are␣dead␣
    !!"))
191                         pass
192                 listValid.append(cmd)
193 
194             elif cmd.startswith("A_PLAY␣"):
195                 cmdtmp = cmd.split('␣')
196 
    self.model.add_character(cmdtmp[1],bool(int(cmdtmp[2])),int(cmdtmp[3]),(int(cmdtmp[4]),
    int(cmdtmp[5])), int(cmdtmp[6]))
197                 listValid.append(cmd)
198 
199             elif cmd.startswith("A_BOMB␣"):
200                 cmdtmp = cmd.split('␣')
201                 self.model.bombs.append(Bomb(self.model.map,
    (int(cmdtmp[1]),int(cmdtmp[2])),int(cmdtmp[3]),int(cmdtmp[4])))
202                 listValid.append(cmd)
203 
204             elif cmd.startswith("DP_BOMB␣"):
205                 cmdtmp = cmd.split('␣')
206                 try:
207                     self.model.drop_bomb(cmdtmp[1],
    int(cmdtmp[2]), int(cmdtmp[3]))
208                 except:
209                     listValid.append(str("MSG␣You␣are␣dead␣!!"))
210                     pass
211                 listValid.append(cmd)
212 
213             elif cmd.startswith("A_FRUIT␣"):
214                 cmdtmp = cmd.split('␣')
215                 self.model.add_fruit(int(cmdtmp[1]),
    (int(cmdtmp[2]), int(cmdtmp[3])))
216                 listValid.append(cmd)
217 
218             elif cmd.startswith("S_LIFE␣"):
219                 cmdtmp = cmd.split('␣')
220                 player = self.model.look(cmdtmp[1])
221                 if player != None :
222                     player.health = int(cmdtmp[2])
223                 else:
224                     listValid.append(str("KILL␣"+cmdtmp[1]))
225                     pass
226 
227             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 = CommandNetwork(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.")
```

```
285
    newSock.sendall(self.cmd.enc_command(str("ERROR command init
    new player, name already use.")))
286                     validNick = False
287                     newSock.close();
288                 if s in self.afk:
289                     Afk=True
290
291             if validNick :
292                 self.socks[newSock]= nick
293                 if not Afk :
294                     self.cmd.model.add_character(nick, False)
295                 else:
296                     for s in self.afk:
297                         if self.socks[s]==nick:
298                             self.afk.pop(s)
299                             self.socks.pop(s)
300                             s.close()
301                             break
302
303                 print("New connection")
304                 print(addr)
305
306                 # envoyer map, fruits, joueurs, bombes
307                 self.initMap(newSock);
308                 self.initFruits(newSock)
309                 self.initBombs(newSock)
310                 self.initCharacters(newSock,Afk)
311                 newSock.sendall(self.cmd.enc_command(str("END ")))
312         else:
313             print ("Error command init new player")
314             newSock.close();
315
316     '''
317     Doit renvoyer aux autres destinataires
318     '''
319     def re_send(self,sockSender, cmd):
320         for sock in self.socks:
321             if sock != self.soc and sock != sockSender:
322                 try :
323                     sock.sendall(self.cmd.enc_command(cmd))
324                 except:
325                     print (self.socks[sock])
326                     print (cmd)
327                     print("Error message not have been sent.")
328
329     '''
330     Initialise les characters à envoyer
331     '''
332     def initCharacters(self, s, afk):
333         for char in self.cmd.model.characters:
334             if (char.nickname == self.socks[s]):
335                 #is_player = true, send for initialization to
    others = false
336                 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))))
337                 if not afk:
338                     self.re_send(s, str("A_PLAY "+char.nickname+"
    "+"0"+" "+str(char.kind)+" "+ str(char.pos[X])+" "+
    str(char.pos[Y])+" "+ str(char.health)))
339             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")
                                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 = CommandNetwork(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:
```

```
453                 self.soc = socket.socket(res[0], res[1]);
454             except:
455                 self.soc = None;
456                 continue;
457             try:
458                 self.soc.connect(res[4]);
459             except:
460                 self.soc.close();
461                 self.soc = None;
462                 continue;
463             print("Connected.\n");
464             break;
465         if self.soc is None:
466             print("Error : can't open connection.\n");
467             sys.exit(1);
468 
469         print ("Connection to server open.")
470         print ("Send request game ...")
471         print()
472         #Connection
473         self.soc.sendall(self.cmd.enc_command(str("CON "
    "+nickname)));
474 
475 
476         #Decode map + objects (fruits, bombs) + players
477         stop = False
478         while (not stop):
479 
480             msg = self.soc.recv(SIZE_BUFFER_NETWORK)
481             if len(msg )<= 0 :
482                 print ("Brutal interruption of the connection "
    during the chargement of the map.")
483                 sys.exit(1)
484 
485             listCmd = self.cmd.dec_command(msg)
486 
487             if (listCmd==None):
488                 stop = True
489                 print ("Unknow command give by the server, maybe "
    it have not the same version.")
490                 sys.exit(1)
491 
492             for c in listCmd:
493                 if c.startswith("END"):
494                     stop = True
495                     break
496 
497 
498 
499     # keyboard events
500 
501     def keyboard_quit(self):
502         print("=> event \"quit\"")
503         if not self.cmd.model.player: return False
504         self.soc.sendall(self.cmd.enc_command(str("QUIT "
    "+self.cmd.model.player.nickname)))
505         sys.exit()
506         return False
507 
508     def keyboard_move_character(self, direction):
509         print("=> event \"keyboard move direction\" "
    {}".format(DIRECTIONS_STR[direction]))
```

```
510
511         if not self.cmd.model.player: return True
512
513         self.soc.sendall(self.cmd.enc_command(str("MOVE␣
    "+self.cmd.model.player.nickname+"␣"+str(direction))));
514
515         #SOLO
516         nickname = self.cmd.model.player.nickname
517         if direction in DIRECTIONS:
518             self.cmd.model.move_character(nickname, direction)
519
520         return True
521
522     def keyboard_drop_bomb(self):
523         print("=>␣event␣\"keyboard␣drop␣bomb\"")
524
525         if not self.cmd.model.player: return True
526
527         self.soc.sendall(self.cmd.enc_command(str("DP_BOMB␣
    "+self.cmd.model.player.nickname+"␣"+str(MAX_RANGE)+"␣
    "+str(COUNTDOWN))));
528
529         #SOLO
530         nickname = self.cmd.model.player.nickname
531         self.cmd.model.drop_bomb(nickname)
532
533         return True
534
535     # time event
536
537     def tick(self, dt):
538         sel = select.select([self.soc], [], [], 0);
539         if sel[0]:
540             for s in sel[0]:
541                 try:
542                     msg = s.recv(SIZE_BUFFER_NETWORK);
543                 except:
544                     print ("Error:␣Server␣has␣been␣disconnected")
545                     s.close();
546                     sys.exit(1)
547
548                 if (len(msg) <= 0):
549                     print ("Error:␣message␣empty,␣server␣has␣been␣
    disconnected")
550                     s.close();
551                     sys.exit(1)
552
553                 listCmd = self.cmd.dec_command(msg)
554                 if (listCmd==None):
555                     print ("Unknow␣command␣give␣by␣the␣server,␣
    maybe␣it␣have␣not␣the␣same␣version.")
556                     sys.exit(1)
557
558         if self.cmd.model.player != None :
559             self.soc.sendall(self.cmd.enc_command(str("S_LIFE␣
    "+str(self.cmd.model.player.nickname)+"␣
    "+str(self.cmd.model.player.health))));
560
561
562         return True
```