Université de Bordeaux
Licence informatique

26 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 le jeu *Bomberman*

Table des matières

Première partie
Preambule

Deuxième partie

Projet réseau

Troisième partie

Annexes

A Moodle

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

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>

```

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

```
                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.")
```

```
284
    newSock.sendall(self.cmd.enc_command(str("ERROR command init
    new player, name already use.")))
285                     validNick = False
286                     newSock.close();
287                 if s in self.afk:
288                     Afk=True
289
290             if validNick :
291                 self.socks[newSock]= nick
292                 if not Afk :
293                     self.cmd.model.add_character(nick, False)
294                 else:
295                     for s in self.afk:
296                         if self.socks[s]==nick:
297                             self.afk.pop(s)
298                             self.socks.pop(s)
299                             s.close()
300                             break
301
302                 print("New connection")
303                 print(addr)
304
305                 # envoyer map, fruits, joueurs, bombes
306                 self.initMap(newSock);
307                 self.initFruits(newSock)
308                 self.initBombs(newSock)
309                 self.initCharacters(newSock,Afk)
310                 newSock.sendall(self.cmd.enc_command(str("END ")))
311         else:
312             print ("Error command init new player")
313             newSock.close();
314
315     '''
316     Doit renvoyer aux autres destinataires
317     '''
318     def re_send(self,sockSender, cmd):
319         for sock in self.socks:
320             if sock != self.soc and sock != sockSender:
321                 try :
322                     sock.sendall(self.cmd.enc_command(cmd))
323                 except:
324                     print (self.socks[sock])
325                     print (cmd)
326                     print("Error message not have been sent.")
327
328     '''
329     Initialise les characters à envoyer
330     '''
331     def initCharacters(self, s, afk):
332         for char in self.cmd.model.characters:
333             if (char.nickname == self.socks[s]):
334                 #is_player = true, send for initialization to
    others = false
335                 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))))
336                 if not afk:
337                     self.re_send(s, str("A_PLAY "+char.nickname+"
    "+"0"+" "+str(char.kind)+" "+ str(char.pos[X])+" "+
    str(char.pos[Y])+" "+ str(char.health)))
338             else:
```

```
339                 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))))
340
341     '''
342     Initialise les fruits à envoyer
343     '''
344     def initFruits(self, s):
345         for fruit in self.cmd.model.fruits:
346             s.sendall(self.cmd.enc_command(str("A_FRUIT␣
    "+str(FRUITS[fruit.kind])+"␣"+ str(fruit.pos[X])+"␣"+
    str(fruit.pos[Y]))))
347         return
348     '''
349     Initialise les bombs à envoyer
350     '''
351     def initBombs(self, s):
352         for bomb in self.cmd.model.bombs:
353             s.sendall(self.cmd.enc_command(str("A_BOMB␣
    "+str(bomb.pos[X])+"␣"+str(bomb.pos[Y])+"␣
    "+str(bomb.max_range)+"␣"+str(bomb.countdown))))
354         return
355
356     '''
357     Initialise la map à envoyer
358     '''
359     def initMap(self, s):
360         if len(sys.argv) == 3:
361             s.sendall(self.cmd.enc_command(str("MAP␣
    "+sys.argv[2])));
362         else:
363             s.sendall(self.cmd.enc_command(str("MAP␣
    "+DEFAULT_MAP)));
364         return
365
366     '''
367     Déconnecte un client et renvoie le nom du joueur à supprimer
368     '''
369     def disconnectClient(self, s):
370         if s in self.socks:
371             nick = self.socks[s]
372             self.cmd.model.quit(nick);
373             s.close()
374             self.socks.pop(s)
375             self.re_send(s, str("KILL␣"+ nick))
376
377
378     # time event
379
380     def tick(self, dt):
381         sel = select.select(self.socks, [], [], 0);
382         if sel[0]:
383             for s in sel[0]:
384                 if s is self.soc:
385                     self.clientConnection(s);
386
387                 elif s in self.socks:
388                     if s not in self.afk:
389                         msg =b""
390                         try:
391                             msg = s.recv(SIZE_BUFFER_NETWORK);
392                         except:
```

```
393                                 print ("Error interuption")
394                                 print("Connection client afk.")
395                                 self.afk[s]=(TIMEOUT+1)*1000-1
396                                 #self.disconnectClient(s)
397                                 break
398 
399                             if (len(msg) <= 0):
400                                 print ("Error message empty.")
401                                 self.afk[s]=(TIMEOUT+1)*1000-1
402                                 #self.disconnectClient(s)
403                                 break
404 
405                             else:
406                                 listCmd = self.cmd.dec_command(msg)
407                                 for cmd in listCmd:
408                                     if cmd.startswith("QUIT"):
409                                         self.disconnectClient(s)
410                                         break
411                                     else:
412                                         self.re_send(s, cmd)
413 
414                             for char in self.cmd.model.characters:
415                                 self.re_send(s ,str("S_LIFE "
    "+str(char.nickname)+" "+str(char.health)));
416                         else:
417                             try:
418                                 msg = s.recv(SIZE_BUFFER_NETWORK);
419                                 self.afk.pop(s)
420 
421                             except:
422                                 self.afk[s]-=dt
423                                 print(int(self.afk[s] / 1000))
424                                 if (self.afk[s]<0):
425                                     print ("timeout connection")
426                                     print (self.socks[s])
427                                     self.afk.pop(s)
428                                     self.disconnectClient(s)
429 
430 
431         return True
432 
433 ##############################################################################
434 #                           NETWORK CLIENT CONTROLLER
                #
435 ##############################################################################
436 
437 class NetworkClientController:
438 
439     def __init__(self, model, host, port, nickname):
440         self.host = host;
441         self.port = port;
442         self.cmd = Command_Network(model,False)
443         self.nickname = nickname;
444         self.soc = None;
445         try:
446             request = socket.getaddrinfo(self.host, self.port, 0,
    socket.SOCK_STREAM);
447         except:
448             print("Error : can't connect to server.\n");
449             sys.exit(1);
450         for res in request:
451             try:
```

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

```
509
510         if not self.cmd.model.player: return True
511
512         self.soc.sendall(self.cmd.enc_command(str("MOVE␣
    "+self.cmd.model.player.nickname+"␣"+str(direction))));
513
514         #SOLO
515         nickname = self.cmd.model.player.nickname
516         if direction in DIRECTIONS:
517             self.cmd.model.move_character(nickname, direction)
518
519         return True
520
521     def keyboard_drop_bomb(self):
522         print("=>␣event␣\"keyboard␣drop␣bomb\"")
523
524         if not self.cmd.model.player: return True
525
526         self.soc.sendall(self.cmd.enc_command(str("DP_BOMB␣
    "+self.cmd.model.player.nickname+"␣"+str(MAX_RANGE)+"␣
    "+str(COUNTDOWN))));
527
528         #SOLO
529         nickname = self.cmd.model.player.nickname
530         self.cmd.model.drop_bomb(nickname)
531
532         return True
533
534     # time event
535
536     def tick(self, dt):
537         sel = select.select([self.soc], [], [], 0);
538         if sel[0]:
539             for s in sel[0]:
540                 try:
541                     msg = s.recv(SIZE_BUFFER_NETWORK);
542                 except:
543                     print ("Error:␣Server␣has␣been␣disconnected")
544                     s.close();
545                     sys.exit(1)
546
547                 if (len(msg) <= 0):
548                     print ("Error:␣message␣empty,␣server␣has␣been␣
    disconnected")
549                     s.close();
550                     sys.exit(1)
551
552                 listCmd = self.cmd.dec_command(msg)
553                 if (listCmd==None):
554                     print ("Unknow␣command␣give␣by␣the␣server,␣
    maybe␣it␣have␣not␣the␣same␣version.")
555                     sys.exit(1)
556
557         if self.cmd.model.player != None :
558             self.soc.sendall(self.cmd.enc_command(str("S_LIFE␣
    "+str(self.cmd.model.player.nickname)+"␣
    "+str(self.cmd.model.player.health))));
559
560
561         return True
```