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 B).

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 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

Nous avons choisi d'utiliser TCP, car ainsi nous évitions de perdre des données en transit nécessaires au bon déroulement du jeu. Cela permettait d'assurer une synchronisation efficace entre les différents copies du jeu.

En ce qui concerne la gestion des connexions, nous avons choisi d'utiliser select au lieu des threads, car du fait du partage de certaines données comme le model, la liste des sockets, etc, on ne voulait pas "bloquer" ces variables et qu'elles soient disponibles de modifications à n'importe quel moment du jeu sans avoir de problème d'accès. Nous étions plus familier aussi de cet outil grâce au mini-chat réseau [1].

Puis l'envoi et la réception des données, nous avons choisi d'utiliser les sendall/recv de la bibliotheque socket (B) en mode non bloquant, car de une, il ne fallait pas bloquer la boucle de jeu pour que tout s'actualise en temps réel, de deux, le sendall permettait de garder un ordre d'envoi chronologique, et nous avons sur les recv gérer quand le buffer recevait plusieurs instructions à la fois.

Ce fut réalisé grâce aux formalismes d'envoi dans la class CommandNetwork, ou on encodait la commande au départ de string en byte array avec enc_command, et on décodait et on altérait le modèle avec dec_command, qui vérifier aussi si la commande existait. Les commandes réseau sont de cette forme :

CMD <arg1> <arg2>

Vous avez la liste des commandes en D.1

1. Travail réalisé durant l'année en réseau.

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.

Le principe de cette algorithme est de vérifier pour chaque bombe posé la position d'arrivé du joueur et de celle-ci pour bloquer les mouvements. Nous avions réfléchi à d'autres implémentations en rajoutant sur la map la bombe posé et de vérifier uniquement sur la case d'arrivé du joueur si il y a une bombe, mais cela demanderait beaucoup d'effort pour un petit jeu et plus de mémoire "inutile" si il y a pas de bombes. Comme en plus du fait du timer des bombes et de la période où on peut plus placer de bombes, le nombre de bombes sur le plateau est limité.

```
1
2     #return true if the aimed case is a bomb
3     def colliderBomb(self, chara, direction, bomb):
4         # move right
5         if direction == DIRECTION_RIGHT:
6             if (chara.pos[X] + 1,chara.pos[Y]) == bomb.pos:
7                 return True
8         # move left
9         elif direction == DIRECTION_LEFT:
10            if (chara.pos[X] - 1,chara.pos[Y]) == bomb.pos:
11                return True
12        # move up
13        elif direction == DIRECTION_UP:
14            if (chara.pos[X],chara.pos[Y] - 1) == bomb.pos:
15                return True
16        # move down
17        elif direction == DIRECTION_DOWN:
18            if (chara.pos[X],chara.pos[Y] + 1) == bomb.pos:
19                return True
20
21        return False;
22
23    # move a character
24    def move_character(self, nickname, direction):
25        character = self.look(nickname)
26        if not character:
27            print("Error: nickname \"{}\" not
      found!".format(nickname))
28            sys.exit(1)
29
30        validBombMove = True
31        for bomb in self.bombs:
32            if(self.colliderBomb(character, direction, bomb)):
33                validBombMove= False
34                break
35
36        if validBombMove :
37            character.move(direction)
38            print("=> move {} \"{}\" at position
      ({},{})".format(DIRECTIONS_STR[direction], nickname,
      character.pos[X], character.pos[Y]))
```

4.2 Gestion des déconnexions

5 Bilan et critique

Troisième partie

Annexes

A Doc python socket

https ://docs.python.org/3/library/socket.html

B Moodle

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

C github professaur

Professeur
https ://github.com/orel33/bomber

Projet
https ://github.com/m21-cerutti/VM_Reseau_L2

D Code Source

D.1 Network.py

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

import socket
import select
import threading
import errno
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
```

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

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

```
            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
                listValid.append(cmd)
```

```
228
229             elif cmd.startswith("KILL ") or cmd.startswith("QUIT
    "):
230                 cmdtmp = cmd.split(' ')
231                 try:
232                     self.model.kill_character(cmdtmp[1]);
233                     print (cmd)
234                 except:
235                     pass
236
237                 listValid.append(cmd)
238
239             elif cmd.startswith("END"):
240                 cmdtmp = cmd.split(' ')
241                 listValid.append(cmd)
242
243             else:
244                 return None
245
246         return listValid;
247
248
249
250
251
252
253
254 ##########################################################################################
255 #                                NETWORK SERVER CONTROLLER
                #
256 ##########################################################################################
257
258 class NetworkServerController:
259
260     def __init__(self, model, port):
261         self.port = port;
262         self.cmd = CommandNetwork(model,True)
263         self.soc = socket.socket(socket.AF_INET6,
    socket.SOCK_STREAM);
264         self.soc.setsockopt(socket.SOL_SOCKET,
    socket.SO_REUSEADDR, 1);
265         self.soc.bind(('', port));
266         self.soc.listen(1);
267         self.socks = {};
268         self.afk={}
269         self.socks[self.soc] = "SERVER";
270
271     '''
272     Connection d'un nouveau client, initialise ses champs
273     '''
274     def clientConnection(self, sockserv):
275         newSock, addr= sockserv.accept()
276         msg = newSock.recv(SIZE_BUFFER_NETWORK)
277
278         listcmd = self.cmd.dec_command(msg)
279
280         if (listcmd!=None and listcmd[0].startswith("CON")):
281             nick= listcmd[0].split(" ")[1]
282             validNick = True
283             Afk = False
284
285             if nick in self.afk:
```

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

                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))))
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 supprime son personnage
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     Déconnecte un client et le rend AFK
379     '''
380     def disconnectAFKClient(self, s):
381         if s in self.socks:
382             nick = self.socks[s]
383             self.afk[nick]=(TIMEOUT+1)*1000-1
384             s.close()
385             self.socks.pop(s)
386             print ("Pass␣to␣AFK")
387             print (nick)
388
389
390
391     # time event
392
393     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 :
                    msg =b""
                    try:
                        msg = s.recv(SIZE_BUFFER_NETWORK);
                    except OSError as e:
                        print(e)
                        self.disconnectAFKClient(s)
                        break

                    if (len(msg) <= 0):
                        print ("Error message empty.")
                        self.disconnectAFKClient(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 nick in self.afk:
            self.afk[nick]-=dt
            #print(int(self.afk[s] / 1000))
            if (self.afk[nick]<0):
                print ("Timeout connection")
                print (nick)
                self.afk.pop(nick)
                self.re_send(self.soc, str("KILL "+ nick))
                break


        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:
                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)))
        return True

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

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