

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  
*Bombberman*

# Table des matières

<b>I</b>	<b>Préambule</b>	<b>2</b>
<b>II</b>	<b>Projet réseau</b>	<b>3</b>
1	Méthode de travail	3
2	Analyse du modèle	3
3	Algorithme et implémentation	3
3.1	Protocoles . . . . .	3
3.2	Choix techniques . . . . .	3
4	Améliorations effectuées	3
4.1	Collisions sur les bombes . . . . .	3
4.2	Gestion des déconnexions . . . . .	4
5	Bilan et critique	4
<b>III</b>	<b>Annexes</b>	<b>5</b>
A	Doc python socket	5
B	Moodle	5
C	github professeur	5
D	Code Source	6
D.1	Network.py . . . . .	6

## 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 *Bombberman* 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 *Bombberman* 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<sup>1</sup>.

Puis l'envoi et la réception des données, nous avons choisi d'utiliser les sendall/recv de la bibliothèque 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 des commandes réseau 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.

### **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 professeur**

Professeur

<https://github.com/orel33/bomber>

Projet

[https://github.com/m21-cerutti/VM\\_Reseau\\_L2](https://github.com/m21-cerutti/VM_Reseau_L2)

## D Code Source

### D.1 Network.py

```
1  # -*- coding: Utf-8 -*-
2  # Author: aurelien.esnard@u-bordeaux.fr
3
4  import socket
5  import select
6  import threading
7  import errno
8  import sys
9  from model import *
10
11  #####
12  #                                     AUXILLARY FUNCTION NETWORK
13  #                                     #
14  #####
15  #Size taken to the socket's buffer
16  SIZE_BUFFER_NETWORK = 2056
17  #Timeout for deconnection afk
18  TIMEOUT = 20
19
20
21  class CommandNetwork:
22
23      def __init__(self, model, isServer):
24          self.model = model;
25          self.isServer = isServer;
26
27      '''
28          #Commands
29          -----
30
31          #End for big transmissions with loops.
32          END
33
34          #Send a message to the client
35          MSG <msg>
36
37          #Send error and close the client
38          ERROR <msg>
39
40          #Connection player
41          CON <nicknamePlayer>
42
43          #Transmit map
44          MAP <namemap>
45
46          #Move player
47          MOVE <nicknamePlayer> <direction>
48
49          #Add player
50          A_PLAY <nicknamePlayer> <isplayer> <kind> <posX> <posY>
51          <health>
52
53          #Add bomb
54          A_BOMB <pos X> <pos Y> <range> <countdown>
55
56          #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

```

```

173         elif cmd.startswith("ERROR_"):
174             cmdtmp = cmd.partition('_')
175             print("ERROR_: "+ cmdtmp[2])
176             sys.exit(1)
177
178         elif cmd.startswith("MAP_"):
179             cmdtmp = cmd.split('_')
180             self.model.load_map(cmdtmp[1])
181             listValid.append(cmd)
182
183         elif cmd.startswith("MOVE_"):
184             cmdtmp = cmd.split('_')
185             nickname = cmdtmp[1]
186             direction = int(cmdtmp[2])
187             if direction in DIRECTIONS:
188                 try:
189                     self.model.move_character(nickname,
190 direction)
191                 except:
192                     listValid.append(str("MSG_You_are_dead_
193 !!"))
194                 pass
195             listValid.append(cmd)
196
197         elif cmd.startswith("A_PLAY_"):
198             cmdtmp = cmd.split('_')
199
200             self.model.add_character(cmdtmp[1], bool(int(cmdtmp[2])), int(cmdtmp[3]), (int(cmdtmp[4]),
201 int(cmdtmp[5])), int(cmdtmp[6]))
202             listValid.append(cmd)
203
204         elif cmd.startswith("A_BOMB_"):
205             cmdtmp = cmd.split('_')
206             self.model.bombs.append(Bomb(self.model.map,
207 (int(cmdtmp[1]), int(cmdtmp[2])), int(cmdtmp[3]), int(cmdtmp[4])))
208             listValid.append(cmd)
209
210         elif cmd.startswith("DP_BOMB_"):
211             cmdtmp = cmd.split('_')
212             try:
213                 self.model.drop_bomb(cmdtmp[1],
214 int(cmdtmp[2]), int(cmdtmp[3]))
215             except:
216                 listValid.append(str("MSG_You_are_dead_!!"))
217             pass
218             listValid.append(cmd)
219
220         elif cmd.startswith("A_FRUIT_"):
221             cmdtmp = cmd.split('_')
222             self.model.add_fruit(int(cmdtmp[1]),
223 (int(cmdtmp[2]), int(cmdtmp[3])))
224             listValid.append(cmd)
225
226         elif cmd.startswith("S_LIFE_"):
227             cmdtmp = cmd.split('_')
228             player = self.model.look(cmdtmp[1])
229             if player != None:
230                 player.health = int(cmdtmp[2])
231             else:
232                 listValid.append(str("KILL_"+cmdtmp[1]))
233             pass
234             listValid.append(cmd)

```

```

228         elif cmd.startswith("KILL") or cmd.startswith("QUIT"):
229             cmdtmp = cmd.split(' ')
230             try:
231                 self.model.kill_character(cmdtmp[1]);
232                 print (cmd)
233             except:
234                 pass
235
236             listValid.append(cmd)
237
238         elif cmd.startswith("END"):
239             cmdtmp = cmd.split(' ')
240             listValid.append(cmd)
241
242         else:
243             return None
244
245         return listValid;
246
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,
264 socket.SOCK_STREAM);
265         self.soc.setsockopt(socket.SOL_SOCKET,
266 socket.SO_REUSEADDR, 1);
267         self.soc.bind((' ', port));
268         self.soc.listen(1);
269         self.socks = {};
270         self.afk={}
271         self.socks[self.soc] = "SERVER";
272
273     '''
274     Connection d'un nouveau client, initialise ses champs
275     '''
276     def clientConnection(self, sockserv):
277         newSock, addr= sockserv.accept()
278         msg = newSock.recv(SIZE_BUFFER_NETWORK)
279
280         listcmd = self.cmd.dec_command(msg)
281
282         if (listcmd!=None and listcmd[0].startswith("CON")):
283             nick= listcmd[0].split(" ")[1]
284             validNick = True
285             Afk = False
286
287             if nick in self.afk:

```

```

286         Afk=True
287     else:
288         for s in self.socks:
289             if self.socks[s]== nick:
290                 print ( "Error_command_init_new_player ,
name_already_use." )
291
292 newSock.sendall( self.cmd.enc_command( str ( "ERROR_command_init_
new_player ,name_already_use." )))
293         validNick = False
294         newSock.close();
295
296     if validNick :
297         self.socks[newSock]= nick
298         if not Afk :
299             self.cmd.model.add_character(nick , False)
300         else:
301             self.afk.pop(nick)
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
313 else:
314     print ( "Error_command_init_new_player" )
315     newSock.close();
316
317 '''
318 Doit renvoyer aux autres destinataires
319 '''
320 def re_send(self,sockSender , cmd):
321     for sock in self.socks:
322         if sock != self.soc and sock != sockSender:
323             try :
324                 sock.sendall( self.cmd.enc_command(cmd))
325             except:
326                 print (self.socks[sock])
327                 print (cmd)
328                 print ( "Error_message_not_have_been_sent." )
329
330 '''
331 Initialise les characters à envoyer
332 '''
333 def initCharacters(self , s, afk):
334     for char in self.cmd.model.characters:
335         if (char.nickname == self.socks[s]):
336             #is_player = true , send for initialization to
others = false
337             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)))
338             if not afk:
339                 self.re_send(s, str ( "A_PLAY_" +char.nickname+"
"+ "0" +" "+ str (char.kind)+" "+ str (char.pos[X])+" "+
str (char.pos[Y])+" "+ str (char.health)))
340             else:
341                 s.sendall( self.cmd.enc_command( str ( "A_PLAY_

```

```

340 "+char.nickname+"␣"+"0"+"␣"+str(char.kind)+"␣"+
341 str(char.pos[X])+"␣"+ str(char.pos[Y])+"␣"+ str(char.health)))
342
343 '''
344 Initialise les fruits à envoyer
345 '''
346 def initFruits(self, s):
347     for fruit in self.cmd.model.fruits:
348         s.sendall(self.cmd.enc_command(str("A_FRUIT␣
349 "+str(FRUIT[fruit.kind])+"␣"+ str(fruit.pos[X])+"␣"+
350 str(fruit.pos[Y]))))
351     return
352
353 '''
354 Initialise les bombs à envoyer
355 '''
356 def initBombs(self, s):
357     for bomb in self.cmd.model.bombs:
358         s.sendall(self.cmd.enc_command(str("A_BOMB␣
359 "+str(bomb.pos[X])+"␣"+str(bomb.pos[Y])+"␣
360 "+str(bomb.max_range)+"␣"+str(bomb.countdown))))
361     return
362
363 '''
364 Initialise la map à envoyer
365 '''
366 def initMap(self, s):
367     if len(sys.argv) == 3:
368         s.sendall(self.cmd.enc_command(str("MAP␣
369 "+sys.argv[2])));
370     else:
371         s.sendall(self.cmd.enc_command(str("MAP␣
372 "+DEFAULT_MAP)));
373     return
374
375 '''
376 Déconnecte un client et supprime son personnage
377 '''
378 def disconnectClient(self, s):
379     if s in self.socks:
380         nick = self.socks[s]
381         self.cmd.model.quit(nick);
382         s.close()
383         self.socks.pop(s)
384         self.re_send(s, str("KILL␣"+ nick))
385
386 '''
387 Déconnecte un client et le rend AFK
388 '''
389 def disconnectAFKClient(self, s):
390     if s in self.socks:
391         nick = self.socks[s]
392         self.afk[nick]=(TIMEOUT+1)*1000-1
393         s.close()
394         self.socks.pop(s)
395         print ("Pass␣to␣AFK")
396         print (nick)
397
398 # time event
399
400 def tick(self, dt):

```

```

394     sel = select.select(self.socks, [], [], 0);
395     if sel[0]:
396         for s in sel[0]:
397             if s is self.soc:
398                 self.clientConnection(s);
399
400             elif s in self.socks :
401                 msg =b""
402                 try:
403                     msg = s.recv(SIZE_BUFFER_NETWORK);
404                 except OSError as e:
405                     print(e)
406                     self.disconnectAFKClient(s)
407                     break
408
409                 if (len(msg) <= 0):
410                     print ("Error_message_empty.")
411                     self.disconnectAFKClient(s)
412                     break
413
414                 else:
415                     listCmd = self.cmd.dec_command(msg)
416                     for cmd in listCmd:
417                         if cmd.startswith("QUIT"):
418                             self.disconnectClient(s)
419                             break
420                         else:
421                             self.re_send(s, cmd)
422
423         for nick in self.afk:
424             self.afk[nick]-=dt
425             #print(int(self.afk[s] / 1000))
426             if (self.afk[nick]<0):
427                 print ("Timeout_connection")
428                 print (nick)
429                 self.afk.pop(nick)
430                 self.re_send(self.soc, str("KILL_"+ nick))
431                 break
432
433     return True
434
435 #####
436 #                                     NETWORK CLIENT CONTROLLER
437 #
438 #####
439
440 class NetworkClientController:
441
442     def __init__(self, model, host, port, nickname):
443         self.host = host;
444         self.port = port;
445         self.cmd = CommandNetwork(model, False)
446         self.nickname = nickname;
447         self.soc = None;
448         try:
449             request = socket.getaddrinfo(self.host, self.port, 0,
450 socket.SOCK_STREAM);
451         except:
452             print("Error: can't connect to server.\n");
453             sys.exit(1);
454         for res in request:

```

```

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

```

```

511         if not self.cmd.model.player: return True
512
513         self.soc.sendall(self.cmd.enc_command(str("MOVE_
514 "+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_
529 "+self.cmd.model.player.nickname+"_"+str(MAX_RANGE)+"_
530 "+str(COUNTDOWN))));
531
532         #SOLO
533         nickname = self.cmd.model.player.nickname
534         self.cmd.model.drop_bomb(nickname)
535
536         return True
537
538     # time event
539
540     def tick(self, dt):
541         sel = select.select([self.soc], [], [], 0);
542         if sel[0]:
543             for s in sel[0]:
544                 try:
545                     msg = s.recv(SIZE_BUFFER_NETWORK);
546                 except OSError as e:
547                     print("Server_closed_connection.")
548                     s.close();
549                     sys.exit()
550
551                 if (len(msg) <= 0):
552                     print("Error:_message_empty,_server_has_been_
553 disconnected")
554                     s.close();
555                     sys.exit(1)
556
557                 listCmd = self.cmd.dec_command(msg)
558                 if (listCmd==None):
559                     print("Unknow_command_give_by_the_server,_
560 maybe_it_have_not_the_same_version.")
561                     sys.exit(1)
562
563                 if self.cmd.model.player != None :
564                     self.soc.sendall(self.cmd.enc_command(str("S_LIFE_
565 "+str(self.cmd.model.player.nickname)+"_
566 "+str(self.cmd.model.player.health))));
567
568         return True

```