

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

I	Preambule	2
II	Projet réseau	3
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 effectues	3
4.1	Collisions sur les bombes	3
4.2	Gestion de déconnexion	3
5	Bilan et critique	3
III	Annexes	4
A	Moodle	4
B	Code Source	5
B.1	Network.py	5

Première partie

Preamble

Dans le cadre de l'enseignement '4TIN401U - Réseaux Info L2' (2017 - 2018) à l'Université de Bordeaux, en semestre 4 de Licence Informatique, nous avons dut adapter le jeu *Bombberman* 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 *Bombberman* fonctionnel en langage Python, avec un rapport fait sur notre travail avant le **le vendredi 27 avril à 23h55**.

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

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

En d'autres termes :

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

Deuxième partie

Projet réseau

1 Méthode de travail

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

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

2 Analyse du modèle

/**/

3 Algorithme et implémentation

3.1 Protocoles

3.2 Choix techniques

4 Améliorations effectues

4.1 Collisions sur les bombes

4.2 Gestion de déconnexion

5 Bilan et critique

Troisième partie

Annexes

A Moodle

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

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

B Code Source

B.1 Network.py

```
1  # -*- coding: Utf-8 -*-
2  # Author: aurelien.esnard@u-bordeaux.fr
3
4  import socket
5  import select
6  import threading
7  import sys
8  from model import *
9
10 #####
11 #                                     AUXILLARY FUNCTION NETWORK
12 #                                     #
13 #####
14 #Size taken to the socket's buffer
15 SIZE_BUFFER_NETWORK = 2056
16 TIMEOUT = 20
17
18
19 class Command_Network:
20
21     def __init__(self, model, isServer):
22         self.model = model;
23         self.isServer = isServer;
24
25
26     '''
27         #Commands
28         -----
29
30         #End for big transmissions with loops.
31         END
32
33         #Send a message to the client
34         MSG <msg>
35
36         #Send error and close the client
37         ERROR <msg>
38
39         #Connection player
40         CON <nicknamePlayer>
41
42         #Transmit map
43         MAP <namemap>
44
45         #Move player
46         MOVE <nicknamePlayer> <direction>
47
48         #Add player
49         A_PLAY <nicknamePlayer> <isplayer> <kind> <posX> <posY>
50         <health>
51
52         #Add bomb
53         A_BOMB <pos X> <pos Y> <range> <countdown>
54
55         #Drop Bomb
56         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] + '_' +
104 cmd[3] + '_' + cmd[4] + '_' + cmd[5] + '_' + cmd[6] + '_' +
105 "\\").encode()
106
107         elif cmd.startswith("MOVE"):
108             cmd = cmd.split('_')
109             return str("MOVE_" + cmd[1] + '_' + cmd[2] + '_' +
110 "\\").encode()
111
112         elif cmd.startswith("A_BOMB"):
113             cmd = cmd.split("_")
114             return str("A_BOMB_" + cmd[1] + '_' + cmd[2] + '_' +
115 cmd[3] + '_' + "\\").encode()
116
117         elif cmd.startswith("DP_BOMB"):
118             cmd = cmd.split("_")

```

```

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

```



```

173         print ("ERROR:_" + cmdtmp[2])
174         sys.exit(1)
175
176     elif cmd.startswith("MAP_"):
177         cmdtmp = cmd.split('_')
178         self.model.load_map(cmdtmp[1])
179         listValid.append(cmd)
180
181     elif cmd.startswith("MOVE_"):
182         cmdtmp = cmd.split('_')
183         nickname = cmdtmp[1]
184         direction = int(cmdtmp[2])
185         if direction in DIRECTIONS:
186             try:
187                 self.model.move_character(nickname,
direction)
188             except:
189                 listValid.append(str("MSG_You_are_dead_
!!"))
190             pass
191             listValid.append(cmd)
192
193     elif cmd.startswith("A_PLAY_"):
194         cmdtmp = cmd.split('_')
195
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_
"):

```

```

227         cmdtmp = cmd.split(' ')
228         try:
229             self.model.kill_character(cmdtmp[1]);
230             print (cmd)
231         except:
232             pass
233
234         listValid.append(cmd)
235
236         elif cmd.startswith("END"):
237             cmdtmp = cmd.split(' ')
238             listValid.append(cmd)
239
240         else:
241             return None
242
243         return listValid;
244
245
246
247
248
249
250
251 #####
252 #                                     NETWORK SERVER CONTROLLER
253 #####
254
255 class NetworkServerController:
256
257     def __init__(self, model, port):
258         self.port = port;
259         self.cmd = Command_Network(model, True)
260         self.soc = socket.socket(socket.AF_INET6,
socket.SOCK_STREAM);
261         self.soc.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1);
262         self.soc.bind((' ', port));
263         self.soc.listen(1);
264         self.socks = {};
265         self.afk={}
266         self.socks[self.soc] = "SERVER";
267
268     '''
269     Connection d'un nouveau client, initialise ses champs
270     '''
271     def clientConnection(self, sockserv):
272         newSock, addr= sockserv.accept()
273         msg = newSock.recv(SIZE_BUFFER_NETWORK)
274
275         listcmd = self.cmd.dec_command(msg)
276
277         if (listcmd!=None and listcmd[0].startswith("CON")):
278             nick= listcmd[0].split(" ")[1]
279             validNick = True
280             Afk = False
281             for s in self.socks:
282                 if self.socks[s]== nick and s not in self.afk:
283                     print ("Error_command_init_new_player, name_
already_use.")
284

```

```

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
336             others = false
337             s.sendall(self.cmd.enc_command(str("A_PLAY_
338 "+char.nickname+"1"+str(char.kind)+" "+
339 str(char.pos[X])+" "+str(char.pos[Y])+" "+str(char.health))))
340             if not afk:
341                 self.re_send(s, str("A_PLAY_"+char.nickname+"
342 "+0"+str(char.kind)+" "+str(char.pos[X])+" "+
343 str(char.pos[Y])+" "+str(char.health)))
344             else:
345                 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 renvoie le nom du joueur à supprimer
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 # time event
388
389 def tick(self, dt):
390     sel = select.select(self.socks, [], [], 0);
391     if sel[0]:
392         for s in sel[0]:
393             if s is self.soc:
394                 self.clientConnection(s);
395
396             elif s in self.socks:
397                 if s not in self.afk:
398                     msg =b""
399                     try:
400                         msg = s.recv(SIZE_BUFFER_NETWORK);
401                     except:
402                         print("Error interruption")

```

```

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("Unknown 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     # keyboard events
498
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(COUIDOWN))));
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([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

```