Rapport pour Projet Réseau TM1A

AMEEUW Vincent CERUTTI Marc

19 avril 2018

Résumé

Ébauche de rapport reseau *Bomberman*

Table des matières

Première partie

Preambule

Vous devez rendre ici deux fichiers :

un rapport PDF (de 5 à 10 pages max, police 12) décrivant votre projet, organisé comme cela : introduction rapide au projet vue d'ensemble fonctionnelle : de ce qui marche parfaitement, de ce qui ne marche pas parfaitement, des bonus et de ce qui reste à faire, ... architecture et implementation : vous prendrez soin de décrire les choix techniques que vous avez effectués (TCP/UDP, select/thread, recv non bloquant, acquittement, ...), ainsi que de décrire votre protocole réseau au moyen d'un formalisme de votre choix (shéma, algorithme en pseudo-langage, automate, ...). bilan du projet une archive ZIP contenant tous les fichiers sources et ressources (images, ...) utiles au bon fonctionnement de votre projet, ainsi qu'un README détaillant comment lancer et jouer avec votre programme.

La date de rendu est fixée au **vendredi 27 avril 23h55**.

Deuxième partie

Projet réseau

Troisième partie

Annexes

A Code Source

A.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


class Command_Network:

    def __init__(self, model, isServer):
        self.model = model;
        self.isServer = isServer;


    '''
        #Commands
        _________
        """
        Finit les transmissions pour les listes d'objets.
        """
        END

        """
        Envoie un message à afficher
        """
        MSG <msg>

        """
        Envoie une erreur à afficher et ferme le client qui le reçoit.
        """
        ERROR <msg>

        """
        Connection d'un joueur avec son nom.
        """
        CON <nicknamePlayer>

        """
        Transmission de la map à charger
        """
        MAP <namemap>

        """
        Déplace le joueur par son nom
        """
        MOVE <nicknamePlayer> <direction>
```

```
57
58          """
59          Ajoute un joueur
60          """
61          #player
62          A_PLAY <nicknamePlayer> <isplayer> <kind> <posX> <posY>
63
64          """
65          Ajoute une bombe
66          """
67          A_BOMB <pos X> <pos Y>
68          #model.bombs.append(Bomb(self.map, character.pos))
69
70          """
71          Drop Bomb par personnage
72          """
73          DP_BOMB <nicknamePlayer>
74
75          """
76          Ajoute un fruit
77          """
78          #fruit
79          A_FRUIT <kind> <pos X> <pos Y>
80
81
82      '''
83      '''
84      Encode les commandes pour l'envoi réseau.
85      En cas de commande inconnu, retourne None.
86      '''
87      def enc_command(self, cmd):
88          cmd.replace('\\','')
89
90          print ("ENC")
91          print (cmd)
92          print ()
93
94          if cmd.startswith("CON"):
95              cmd = cmd.split("␣")
96              return str("CON␣" + cmd[1] + "␣\\").encode()
97
98          elif cmd.startswith("MSG"):
99              cmd = cmd.partition("␣")
100             return str("MSG␣" + cmd[2] + "␣\\").encode()
101
102         elif cmd.startswith("ERROR"):
103             cmd = cmd.partition("␣")
104             return str("ERROR␣" + cmd[2] + "␣\\").encode()
105
106         elif cmd.startswith("MAP"):
107             cmd =cmd.split("␣")
108             return str("MAP␣" + cmd[1]  +"␣\\").encode()
109
110         elif cmd.startswith("A_PLAY"):
111             cmd =cmd.split("␣")
112             return str("A_PLAY␣" + cmd[1] + '␣' + cmd[2] + '␣' +
    cmd[3] +  '␣' + cmd[4] + '␣' + cmd[5] + "␣\\").encode()
113
114         elif cmd.startswith("MOVE"):
115             cmd = cmd.split('␣')
116             return str("MOVE␣" + cmd[1] + '␣' + cmd[2] + "␣
    \\").encode()
```

```
117
118         elif cmd.startswith("A_BOMB"):
119             cmd =cmd.split("␣")
120             return str("A_BOMB␣" + cmd[1] + '␣' + cmd[2]  + "␣
    \\").encode()
121
122         elif cmd.startswith("DP_BOMB"):
123             cmd =cmd.split("␣")
124             return str("DP_BOMB␣" + cmd[1] + "␣\\").encode()
125
126         elif cmd.startswith("A_FRUIT"):
127             cmd =cmd.split("␣")
128             return str("A_FRUIT␣" + cmd[1] + '␣' + cmd[2]  + '␣' +
    cmd[3]  +"␣\\").encode()
129
130         elif cmd.startswith("END"):
131             cmd =cmd.split("␣")
132             return str("END␣" + "\\").encode()
133
134         return None;
135
136     '''
137     Decode les commandes.
138     Adapte le modèle et renvoi une liste de string correspondant
    aux commandes.
139     Return None en cas de commandes inconnus.
140     '''
141     def dec_command(self, msg):
142
143         listCmds = msg.decode()
144         listCmds = listCmds.split('\\')
145         print ("BUFFER")
146         print (listCmds)
147
148         listValid =[]
149
150         while (listCmds != [] and listCmds[0] != ''):
151
152             cmd = listCmds[0]
153             cmd = cmd.replace ('\\','␣')
154             print ("DEC")
155             print (cmd)
156             print ()
157             del listCmds[0]
158
159             if cmd.startswith("CON␣"):
160                 cmdtmp = cmd.split('␣')
161                 listValid.append(cmd)
162
163             elif cmd.startswith("MSG␣"):
164                 cmdtmp = cmd.partition('␣')
165                 print (cmdtmp[2])
166                 listValid.append(cmd)
167
168             elif cmd.startswith("ERROR␣"):
169                 cmdtmp = cmd.partition('␣')
170                 print ("ERROR␣:␣"+ cmdtmp[2])
171                 sys.exit(1)
172
173             elif cmd.startswith("MAP␣"):
174                 cmdtmp = cmd.split('␣')
175                 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:
                    self.model.move_character(nickname, direction)
                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])))
                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]))))
                listValid.append(cmd)

            elif cmd.startswith("DP_BOMB "):
                cmdtmp = cmd.split(' ')
                nickname = cmdtmp[1]
                self.model.drop_bomb(nickname)
                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("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);
```

```
232         self.soc.setsockopt(socket.SOL_SOCKET,
    socket.SO_REUSEADDR, 1);
233         self.soc.bind(('', port));
234         self.soc.listen(1);
235         self.socks = {};
236         self.socks[self.soc] = "SERVER";
237 
238     '''
239     Connection d'un nouveau client, initialise ses champs
240     '''
241     def clientConnection(self, sockserv):
242         newSock, addr= sockserv.accept()
243         msg = newSock.recv(SIZE_BUFFER_NETWORK)
244 
245         listcmd = self.cmd.dec_command(msg)
246 
247         if (listcmd!=None and listcmd[0].startswith("CON")):
248             nick= listcmd[0].split(" ")[1]
249             validNick = True
250             for s in self.socks:
251                 if self.socks[s]== nick:
252                     print ("Error command init new player, name
    already use.")
253                     newSock.send(self.cmd.enc_command(str("ERROR
    command init new player, name already use.")))
254                     validNick = False
255                     newSock.close();
256 
257             if validNick :
258                 self.socks[newSock]= nick
259                 self.cmd.model.add_character(nick, False)
260                 print("New connection")
261                 print(addr)
262 
263                 # envoyer map, fruits, joueurs, bombes
264                 self.initMap(newSock);
265                 self.initFruits(newSock)
266                 self.initBombs(newSock)
267                 self.initCharacters(newSock)
268                 newSock.send(self.cmd.enc_command(str("END ")))
269         else:
270             print ("Error command init new player")
271             newSock.close();
272 
273     '''
274     Doit renvoyer aux autres destinataires
275     '''
276     def re_send(self,sockSender, cmd):
277         for sock in self.socks:
278             if sock != self.soc and sock != sockSender:
279                 sock.sendall(self.cmd.enc_command(cmd))
280 
281     '''
282     Initialise les characters à envoyer
283     '''
284     def initCharacters(self, s):
285         for char in self.cmd.model.characters:
286             if (char.nickname == self.socks[s]):
287                 #is_player = true, send for initialization to
    others = false
288                 s.send(self.cmd.enc_command(str("A_PLAY
    "+char.nickname+" "+"1"+" "+str(char.kind)+" "+
```

```
    str(char.pos[X])+" "+ str(char.pos[Y]))))
289                 self.re_send(s, str("A_PLAY "+char.nickname+" "+"0"+" "+str(char.kind)+" "+ str(char.pos[X])+" "+str(char.pos[Y])))
290             else:
291                 s.send(self.cmd.enc_command(str("A_PLAY "+char.nickname+" "+"0"+" "+str(char.kind)+" "+str(char.pos[X])+" "+ str(char.pos[Y]))))
292 
293     '''
294     Initialise les fruits à envoyer
295     '''
296     def initFruits(self, s):
297         for fruit in self.cmd.model.fruits:
298             s.send(self.cmd.enc_command(str("A_FRUIT "+str(FRUITS[fruit.kind])+" "+ str(fruit.pos[X])+" "+str(fruit.pos[Y]))))
299         return
300     '''
301     Initialise les bombs à envoyer
302     '''
303     def initBombs(self, s):
304         for bomb in self.cmd.model.bombs:
305             s.send(self.cmd.enc_command(str("A_BOMB "+bomb.pos[X]+" "+bomb.pos[Y])));
306         return
307 
308     '''
309     Initialise la map à envoyer
310     '''
311     def initMap(self, s):
312         if len(sys.argv) == 3:
313             s.sendall(self.cmd.enc_command(str("MAP "+sys.argv[2])));
314         else:
315             s.sendall(self.cmd.enc_command(str("MAP "+DEFAULT_MAP)));
316         return
317 
318     '''
319     Déconnecte un client
320     '''
321     def disconnectClient(self, s):
322         self.cmd.model.quit(self.socks[s]);
323         del self.socks[s];
324         s.close()
325 
326     # time event
327 
328     def tick(self, dt):
329         sel = select.select(self.socks, [], [], 0);
330         if sel[0]:
331             for s in sel[0]:
332                 if s is self.soc:
333                     self.clientConnection(s);
334                 else:
335                     msg = s.recv(SIZE_BUFFER_NETWORK);
336                     listCmd = self.cmd.dec_command(msg)
337                     for cmd in listCmd:
338                         self.re_send(s, cmd)
339                     if (len(msg) <= 0):
340                         self.disconnectClient(s);
```

```
341
342         return True
343
344 ##############################################################################
345 #                                NETWORK CLIENT CONTROLLER                   #
346 ##############################################################################
347
348 class NetworkClientController:
349
350     def __init__(self, model, host, port, nickname):
351         self.host = host;
352         self.port = port;
353         self.cmd = Command_Network(model,False)
354         self.nickname = nickname;
355         self.soc = None;
356         try:
357             request = socket.getaddrinfo(self.host, self.port, 0,
    socket.SOCK_STREAM);
358         except:
359             print("Error : can't connect to server.\n");
360             sys.exit(1);
361         for res in request:
362             try:
363                 self.soc = socket.socket(res[0], res[1]);
364             except:
365                 self.soc = None;
366                 continue;
367             try:
368                 self.soc.connect(res[4]);
369             except:
370                 self.soc.close();
371                 self.soc = None;
372                 continue;
373             print("Connected.\n");
374             break;
375         if self.soc is None:
376             print("Error : can't open connection.\n");
377             sys.exit(1);
378
379         #Connection
380         self.soc.send(self.cmd.enc_command(str("CON "+nickname)));
381
382         #Decode map + objects (fruits, bombs) + players
383         stop = False
384         while (not stop):
385
386             msg = self.soc.recv(SIZE_BUFFER_NETWORK)
387             if len(msg )<= 0 :
388                 print ("Brutal interruption of the connection.")
389                 sys.exit(1)
390
391             listCmd = self.cmd.dec_command(msg)
392
393             if (listCmd==None):
394                 stop = True
395                 print ("Unknow command give by the server, maybe
    it have not the same version.")
396                 sys.exit(1)
397
398             for c in listCmd:
399                 if c.startswith("END"):
```

```
                    stop = True
                    break



    # keyboard events

    def keyboard_quit(self):
        print("=> event \"quit\"")
        return False

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

        self.soc.send(self.cmd.enc_command(str("MOVE "+self.cmd.model.player.nickname+" "+str(direction))));

        #SOLO
        if not self.cmd.model.player: return True
        nickname = self.cmd.model.player.nickname
        if direction in DIRECTIONS:
            self.cmd.model.move_character(nickname, direction)

        return True

    def keyboard_drop_bomb(self):
        print("=> event \"keyboard drop bomb\"")

        self.soc.send(self.cmd.enc_command(str("DP_BOMB "+self.cmd.model.player.nickname)));

        #SOLO
        if not self.cmd.model.player: return True
        nickname = self.cmd.model.player.nickname
        self.cmd.model.drop_bomb(nickname)

        return True

    # time event

    def tick(self, dt):
        sel = select.select([self.soc], [], [], 0);
        if sel[0]:
            for s in sel[0]:
                msg = s.recv(SIZE_BUFFER_NETWORK);

                if (len(msg) <= 0):
                    print ("Error: Server has been disconnected")
                    s.close();
                    sys.exit(1)

                listCmd = self.cmd.dec_command(msg)


        return True
```