université
de **BORDEAUX**

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 multijoueur (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]. **CMD <arg1> <arg2>**\\

Vous avez la liste des commandes en D.1

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,

---

1. Travail réalisé durant l'année en réseau.

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.

```python
#return true if the aimed case is a bomb
def colliderBomb(self, chara, direction, bomb):
    # move right
    if direction == DIRECTION_RIGHT:
        if (chara.pos[X] + 1,chara.pos[Y]) == bomb.pos:
            return True
    # move left
    elif direction == DIRECTION_LEFT:
        if (chara.pos[X] - 1,chara.pos[Y]) == bomb.pos:
            return True
    # move up
    elif direction == DIRECTION_UP:
        if (chara.pos[X],chara.pos[Y] - 1) == bomb.pos:
            return True
    # move down
    elif direction == DIRECTION_DOWN:
        if (chara.pos[X],chara.pos[Y] + 1) == bomb.pos:
            return True

    return False;

# move a character
def move_character(self, nickname, direction):
    character = self.look(nickname)
    if not character:
        print("Error:␣nickname␣\"{}\"␣not␣
found!".format(nickname))
        sys.exit(1)

    validBombMove = True
    for bomb in self.bombs:
        if(self.colliderBomb(character, direction, bomb)):
            validBombMove= False
            break

    if validBombMove :
        character.move(direction)
        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

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

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

D Code Source

D.1 Network.py

```python
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     ' ' '
29         #Commands
30         _____
31
32         #End for big transmissions with loops .
33         END
34
35         #Send a message to the client
36         MSG <msg>
37
38         #Send error and close the client
39         ERROR <msg>
40
41         #Connection player
42         CON <nicknamePlayer>
43
44         #Transmit map
45         MAP <namemap>
46
47         #Move player
48         MOVE <nicknamePlayer> <direction>
49
50         #Add player
51         A_PLAY <nicknamePlayer> <isplayer> <kind> <posX> <posY>
    <health>
52
53         #Add bomb
54         A_BOMB <pos X> <pos Y> <range> <countdown>
55
56         #Drop Bomb
```

6

```python
           DP_BOMB <nicknamePlayer> <range> <countdown>

           #Add fruit
           A_FRUIT <kind> <pos X> <pos Y>

           #Synchronisation of life
           S_LIFE <nicknamePlayer> <health>

           #Kill player
           KILL <nicknamePlayer>

           #Disconnection of the client
           QUIT <nicknamePlayer>

           #TOADD
           -send map


      '''
      '''
      Encode les commandes pour l'envoi réseau.
      En cas de commande inconnu, retourne None.
      '''
      def enc_command(self, cmd):
          cmd.replace('\\','')

          #print ("ENC")
          #print (cmd)
          #print ()

          if cmd.startswith("CON"):
              cmd = cmd.split(" ")
              return str("CON " + cmd[1] + " \\").encode()

          elif cmd.startswith("MSG"):
              cmd = cmd.partition(" ")
              return str("MSG " + cmd[2] + " \\").encode()

          elif cmd.startswith("ERROR"):
              cmd = cmd.partition(" ")
              return str("ERROR " + cmd[2] + " \\").encode()

          elif cmd.startswith("MAP"):
              cmd =cmd.split(" ")
              return str("MAP " + cmd[1]  +" \\").encode()

          elif cmd.startswith("A_PLAY"):
              cmd =cmd.split(" ")
              return str("A_PLAY " + cmd[1] + ' ' + cmd[2] + ' ' +
      cmd[3] +  ' ' + cmd[4] + ' ' + cmd[5]+ ' ' + cmd[6] + " 
      \\").encode()

          elif cmd.startswith("MOVE"):
              cmd = cmd.split(' ')
              return str("MOVE " + cmd[1] + ' ' + cmd[2] + " 
      \\").encode()

          elif cmd.startswith("A_BOMB"):
              cmd =cmd.split(" ")
              return str("A_BOMB " + cmd[1] + ' ' + cmd[2]  + ' ' +
      cmd[3]+ " \\").encode()
```

```python
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
```

```python
            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)
```

9

```
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:
```

```python
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.")

        newSock.sendall(self.cmd.enc_command(str("ERROR command init 
     new player, name already use.")))
292                            validNick = False
293                            newSock.close();

295                if validNick :
296                    self.socks[newSock]= nick
297                    if not Afk :
298                        self.cmd.model.add_character(nick, False)
299                    else:
300                        self.afk.pop(nick)

302                    print("New connection")
303                    print(addr)

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();

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.")

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 
```

```python
        "+char.nickname+" "+"0"+" "+str(char.kind)+" "+
        str(char.pos[X])+" "+ str(char.pos[Y])+" "+ str(char.health))))
        '''
        Initialise les fruits à envoyer
        '''
        def initFruits(self, s):
            for fruit in self.cmd.model.fruits:
                s.sendall(self.cmd.enc_command(str("A_FRUIT
        "+str(FRUITS[fruit.kind])+" "+ str(fruit.pos[X])+" "+
        str(fruit.pos[Y]))))
            return
        '''
        Initialise les bombs à envoyer
        '''
        def initBombs(self, s):
            for bomb in self.cmd.model.bombs:
                s.sendall(self.cmd.enc_command(str("A_BOMB
        "+str(bomb.pos[X])+" "+str(bomb.pos[Y])+"
        "+str(bomb.max_range)+" "+str(bomb.countdown))))
            return


        '''
        Initialise la map à envoyer
        '''
        def initMap(self, s):
            if len(sys.argv) == 3:
                s.sendall(self.cmd.enc_command(str("MAP
        "+sys.argv[2])));
            else:
                s.sendall(self.cmd.enc_command(str("MAP
        "+DEFAULT_MAP)));
            return


        '''
        Déconnecte un client et supprime son personnage
        '''
        def disconnectClient(self, s):
            if s in self.socks:
                nick = self.socks[s]
                self.cmd.model.quit(nick);
                s.close()
                self.socks.pop(s)
                self.re_send(s, str("KILL "+ nick))

        '''
        Déconnecte un client et le rend AFK
        '''
        def disconnectAFKClient(self, s):
            if s in self.socks:
                nick = self.socks[s]
                self.afk[nick]=(TIMEOUT+1)*1000-1
                s.close()
                self.socks.pop(s)
                print ("Pass to AFK")
                print (nick)



        # time event

        def tick(self, dt):
```

12

```
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
434            return True
435
436 ###############################################################################
437 #                              NETWORK CLIENT CONTROLLER                      #
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,
     socket.SOCK_STREAM);
450         except:
451             print("Error : can't connect to server.\n");
452             sys.exit(1);
453         for res in request:
```

13

```python
            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]))
```

14

```python
            if not self.cmd.model.player: return True

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

            #SOLO
            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\"")

            if not self.cmd.model.player: return True

            self.soc.sendall(self.cmd.enc_command(str("DP_BOMB
        "+self.cmd.model.player.nickname+" "+str(MAX_RANGE)+"
        "+str(COUNTDOWN))));

            #SOLO
            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]:
                    try:
                        msg = s.recv(SIZE_BUFFER_NETWORK);
                    except OSError as e:
                        print ("Server closed connection.")
                        s.close();
                        sys.exit()

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

                    listCmd = self.cmd.dec_command(msg)
                    if (listCmd==None):
                        print ("Unknow command give by the server,
        maybe it have not the same version.")
                        sys.exit(1)

            if self.cmd.model.player != None :
                self.soc.sendall(self.cmd.enc_command(str("S_LIFE
        "+str(self.cmd.model.player.nickname)+"
        "+str(self.cmd.model.player.health))));


            return True
```

15

D.2   Model.py

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

import random
import sys

###################################################################################
#                                         MODEL
#                    #
###################################################################################

### Constants ###

# position / direction
X = 0
Y = 1
DIRECTION_LEFT = 0
DIRECTION_RIGHT = 1
DIRECTION_UP = 2
DIRECTION_DOWN = 3
DIRECTIONS = [DIRECTION_LEFT, DIRECTION_RIGHT, DIRECTION_UP,
    DIRECTION_DOWN]
DIRECTIONS_STR = ["left", "right", "up", "down"]

# map
WALLS = ('w', 'x', 'z')
BACKGROUNDS = ('0', '1', '2')
DEFAULT_MAP = "maps/map0"

# fruit
BANANA = 0
CHERRY = 1
FRUITS = [BANANA, CHERRY]
FRUITS_STR = ["banana", "cherry"]

# character
DK = 0
ZELDA = 1
BATMAN = 2
CHARACTERS = [DK, ZELDA, BATMAN]
CHARACTERS_STR = ["dk", "zelda", "batman"]
HEALTH = 50
MAX_RANGE = 5
COUNTDOWN = 5
IMMUNITY = 1500 # in ms
DISARMED = 2000 # in ms

### Class Map ###

class Map:
    def __init__(self):
        self.array = []
        self.width = 0
        self.height = 0

    def load(self, filename):
        with open(filename, "r") as _file:
            _array = []
            for row in _file:
                _row = []
```

16

```python
59                    for square in row:
60                        if square != '\n':
61                            _row.append(square)
62                    _array.append(_row)
63                self.array = _array
64                self.height = len(self.array)
65                self.width = len(self.array[0])
66
67        def random(self):
68            while True:
69                x = random.randint(0, self.width-1)
70                y = random.randint(0, self.height-1)
71                if self.array[y][x] in BACKGROUNDS:
72                    break
73            return (x,y)
74
75    ### Class Fruit ###
76
77    class Fruit:
78        def __init__(self, kind, m, pos):
79            self.map = m
80            self.pos = pos
81            self.kind = kind
82
83    ### Class Bomb ###
84
85    class Bomb:
86        def __init__(self, m, pos, range_bomb = MAX_RANGE, countdown =
        COUNTDOWN):
87            self.map = m
88            self.pos = pos
89            self.max_range = range_bomb
90            self.countdown = countdown
91            self.time_to_explode = (countdown+1)*1000-1 # in ms
92            # compute bomb range
93            for xmax in range(self.pos[X],
        self.pos[X]+self.max_range+1):
94                if xmax >= m.width or
        self.map.array[self.pos[Y]][xmax] not in BACKGROUNDS: break
95            for ymax in range(self.pos[Y],
        self.pos[Y]+self.max_range+1):
96                if ymax >= m.height or
        self.map.array[ymax][self.pos[X]] not in BACKGROUNDS: break
97            for xmin in range(self.pos[X],
        self.pos[X]-self.max_range-1, -1):
98                if xmin < 0 or self.map.array[self.pos[Y]][xmin] not
        in BACKGROUNDS: break
99            for ymin in range(self.pos[Y],
        self.pos[Y]-self.max_range-1, -1):
100               if ymin < 0 or self.map.array[ymin][self.pos[X]] not
        in BACKGROUNDS: break
101           self.range = [xmin+1, xmax-1, ymin+1, ymax-1]
102
103       def tick(self, dt):
104           # subtract the passed time `dt` from the timer each frame
105           if self.time_to_explode >= 0:
106               self.time_to_explode -= dt
107               self.countdown = int(self.time_to_explode / 1000)
108           else:
109               self.countdown = -1
110
111   ### Class Character ###
```

```python
112
113  class Character:
114      def __init__(self, nickname, kind, m, pos, health = HEALTH):
115          self.map = m
116          self.kind = kind
117          self.health = health
118          self.immunity = 0 # the character gets immunity against
         bomb during this time (in ms)
119          self.disarmed = 0 # the character cannot drop a bomb
         during this time (in ms)
120          self.nickname = nickname
121          self.pos = pos
122          self.direction = DIRECTION_RIGHT
123
124      def move(self, direction):
125          # move right
126          if direction == DIRECTION_RIGHT:
127              if self.pos[X] < (self.map.width - 1):
128                  if self.map.array[self.pos[Y]][self.pos[X] + 1]
         not in WALLS:
129                      self.pos = (self.pos[X]+1, self.pos[Y])
130              self.direction = DIRECTION_RIGHT
131          # move left
132          elif direction == DIRECTION_LEFT:
133              if self.pos[X] > 0:
134                  if self.map.array[self.pos[Y]][self.pos[X] - 1]
         not in WALLS:
135                      self.pos = (self.pos[X]-1, self.pos[Y])
136              self.direction = DIRECTION_LEFT
137          # move up
138          elif direction == DIRECTION_UP:
139              if self.pos[Y] > 0:
140                  if self.map.array[self.pos[Y] - 1][self.pos[X]]
         not in WALLS:
141                      self.pos = (self.pos[X], self.pos[Y]-1)
142              self.direction = DIRECTION_UP
143          # move down
144          elif direction == DIRECTION_DOWN:
145              if self.pos[Y] < (self.map.height - 1):
146                  if self.map.array[self.pos[Y] + 1][self.pos[X]]
         not in WALLS:
147                      self.pos = (self.pos[X], self.pos[Y]+1)
148              self.direction = DIRECTION_DOWN
149
150      def eat(self, fruit):
151          if fruit.pos[X] == self.pos[X] and fruit.pos[Y] ==
         self.pos[Y]:
152              self.health += 10
153              print("{}\'s health: {}".format(self.nickname,
         self.health))
154              return True
155          return False
156
157      def tick(self, dt):
158          # subtract the passed time `dt` from the timer each frame
159          if self.immunity > 0: self.immunity -= dt
160          else: self.immunity = 0
161          if self.disarmed > 0: self.disarmed -= dt
162          else: self.disarmed = 0
163
164      def explosion(self, bomb):
165          if bomb.countdown != 0: return False
```

```python
166             if self.immunity > 0: return False
167             horizontal = (self.pos[Y] == bomb.pos[Y] and self.pos[X]
        >= bomb.range[DIRECTION_LEFT] and self.pos[X] <=
        bomb.range[DIRECTION_RIGHT])
168             vertical = (self.pos[X] == bomb.pos[X] and self.pos[Y] >=
        bomb.range[DIRECTION_UP] and self.pos[Y] <=
        bomb.range[DIRECTION_DOWN])
169             if ( horizontal or vertical ):
170                 self.health -= 10
171                 self.immunity = IMMUNITY
172                 print("{}\'s health: {}".format(self.nickname,
        self.health))
173             if self.health <= 0:
174                 print("{} is dead!".format(self.nickname))
175                 return True
176             return False
177
178 ### Class Model ###
179
180 class Model:
181
182     # initialize model
183     def __init__(self):
184         self.map = Map()
185         self.characters = []
186         self.fruits = []
187         self.bombs = []
188         self.player = None
189
190     # look for a character, return None if not found
191     def look(self, nickname):
192         #
        https://stackoverflow.com/questions/9542738/python-find-in-list
193         character = next( (c for c in self.characters if
        (c.nickname == nickname)), None) # first occurence
194         return character
195
196     # load map from file
197     def load_map(self, filename):
198         self.map.load(filename)
199         print("=> load map \"{}\" of size {}x{}".format(filename,
        self.map.width, self.map.height))
200
201     # kill a character
202     def kill_character(self, nickname):
203         character = self.look(nickname)
204         if not character:
205             print("Error: nickname {} not found!".format(nickname))
206             sys.exit(1)
207         self.characters.remove(character)
208         if self.player == character: self.player = None
209         print("=> kill \"{}\"".format(nickname))
210         return character
211
212     # quit game
213     def quit(self, nickname = None):
214         cont = True
215         if self.player and self.player.nickname == nickname:
216             cont = False
217         character = self.look(nickname)
218         if character: self.kill_character(nickname)
219         print("=> quit \"{}\"".format(nickname))
```

```python
220            return cont
221
222        # add a new fruit
223        def add_fruit(self, kind = None, pos = None):
224            if pos is None: pos = self.map.random()
225            if kind is None: kind = random.choice(FRUITS)
226            self.fruits.append(Fruit(kind, self.map, pos))
227            print("=>_add_fruit_({})_at_position_
        ({},{})".format(FRUITS_STR[kind], pos[X], pos[Y]))
228
229        # add a new character
230        def add_character(self, nickname, isplayer = False, kind =
        None, pos = None, health = None):
231            character = self.look(nickname)
232            if character:
233                print("Error:_nickname_\"{}\"_already_
        used!".format(nickname))
234                sys.exit(1)
235            if pos is None: pos = self.map.random()
236            if kind is None: kind = random.choice(CHARACTERS)
237            if health is None: health = HEALTH
238            character = Character(nickname, kind, self.map, pos,
        health)
239            print("=>_add_character_\"{}\"_({})_as_position_
        ({},{})".format(nickname, CHARACTERS_STR[kind], pos[X],
        pos[Y]))
240            self.characters.append(character)
241            if isplayer: self.player = character
242            return character
243
244        # drop a bomb
245        def drop_bomb(self, nickname, range_bomb = None, countdown =
        None):
246            character = self.look(nickname)
247            if not character:
248                print("Error:_nickname_\"{}\"_not_
        found!".format(nickname))
249                sys.exit(1)
250            if character.disarmed == 0:
251                if range_bomb is None: range_bomb = MAX_RANGE
252                if countdown is None: countdown = COUNTDOWN
253                self.bombs.append(Bomb(self.map, character.pos,
        range_bomb, countdown))
254                character.disarmed = DISARMED
255            print("=>_drop_bomb_at_position_
        ({},{})".format(character.pos[X], character.pos[Y]))
256
257        #return true if the aimed case is a bomb
258        def colliderBomb(self, chara, direction, bomb):
259            # move right
260            if direction == DIRECTION_RIGHT:
261                if (chara.pos[X] + 1,chara.pos[Y]) == bomb.pos:
262                    return True
263            # move left
264            elif direction == DIRECTION_LEFT:
265                if (chara.pos[X] - 1,chara.pos[Y]) == bomb.pos:
266                    return True
267            # move up
268            elif direction == DIRECTION_UP:
269                if (chara.pos[X],chara.pos[Y] - 1) == bomb.pos:
270                    return True
271            # move down
```

```python
272            elif direction == DIRECTION_DOWN:
273                if (chara.pos[X], chara.pos[Y] + 1) == bomb.pos:
274                    return True
275
276            return False;
277
278        # move a character
279        def move_character(self, nickname, direction):
280            character = self.look(nickname)
281            if not character:
282                print("Error:␣nickname␣\"{}\"␣not␣
       found!".format(nickname))
283                sys.exit(1)
284
285            validBombMove = True
286            for bomb in self.bombs:
287                if(self.colliderBomb(character, direction, bomb)):
288                    validBombMove= False
289                    break
290
291            if validBombMove :
292                character.move(direction)
293                print("=>␣move␣{}␣\"{}\"␣at␣position␣
       ({},{})".format(DIRECTIONS_STR[direction], nickname,
       character.pos[X], character.pos[Y]))
294
295        # update model at each clock tick
296        def tick(self, dt):
297            # update bombs (and remove it)
298            for bomb in self.bombs:
299                bomb.tick(dt)
300                if bomb.countdown == -1:
301                    self.bombs.remove(bomb)
302
303            # update characters and eat fruits
304            for character in self.characters:
305                character.tick(dt)
306                for fruit in self.fruits:
307                    if character.eat(fruit):
308                        self.fruits.remove(fruit)
309
310            # update characters after bomb explosion
311            for bomb in self.bombs:
312                for character in self.characters:
313                    if character.explosion(bomb):
314                        self.kill_character(character.nickname)
```

D.3 Bomber_client.py

```python
#!/usr/bin/env python3
# -*- coding: Utf-8 -*
# Author: aurelien.esnard@u-bordeaux.fr

from model import *
from view import *
from keyboard import *
from network import *
import sys
import pygame

### python version ###
print("python version: {}.{}.{}".format(sys.version_info[0],
    sys.version_info[1], sys.version_info[2]))
print("pygame version: ", pygame.version.ver)

################################################################################
#                                    MAIN                                      #
################################################################################

# parse arguments
if len(sys.argv) != 4:
    print("Usage: {} host port nickname".format(sys.argv[0]))
    sys.exit()
host = sys.argv[1]
port = int(sys.argv[2])
nickname = sys.argv[3]

# initialization
pygame.display.init()
pygame.font.init()
clock = pygame.time.Clock()
model = Model()
client = NetworkClientController(model, host, port, nickname)
view = GraphicView(model, nickname)
kb = KeyboardController(client)

# main loop
while True:
    # make sure game doesn't run at more than FPS frames per second
    dt = clock.tick(FPS)
    if not kb.tick(dt): break
    if not client.tick(dt): break
    model.tick(dt)
    view.tick(dt)

# quit
print("Game Over!")
pygame.quit()
```

D.4   Bomber_server.py

```python
#!/usr/bin/env python3
# -*- coding: Utf-8 -*
# Author: aurelien.esnard@u-bordeaux.fr

from model import *
from view import *
from network import *
import sys
import pygame

### python version ###
print("python version: {}.{}.{}".format(sys.version_info[0],
    sys.version_info[1], sys.version_info[2]))
print("pygame version: ", pygame.version.ver)


##########################################################################
#                              MAIN                                      #
##########################################################################

# parse arguments
if len(sys.argv) == 2:
    port = int(sys.argv[1])
    map_file = DEFAULT_MAP
elif len(sys.argv) == 3:
    port = int(sys.argv[1])
    map_file = sys.argv[2]
else:
    print("Usage: {} port [map_file]".format(sys.argv[0]))
    sys.exit()

# initialization
pygame.display.init()
pygame.font.init()
clock = pygame.time.Clock()
model = Model()
model.load_map(map_file)
for _ in range(10): model.add_fruit()
server = NetworkServerController(model, port)
# view = GraphicView(model, "server")

# main loop
while True:
    # make sure game doesn't run at more than FPS frames per second
    dt = clock.tick(FPS)
    server.tick(dt)
    model.tick(dt)
    # view.tick(dt)

# quit
print("Game Over!")
pygame.quit()
```

D.5 Code nécessaire à la compréhension du jeu

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

import pygame
from model import *

##########################################################################
#                          KEYBOARD CONTROLLER
#                    #
##########################################################################


### Class KeyboardController ###

class KeyboardController:

    def __init__(self, evm):
        self.evm = evm
        pygame.key.set_repeat(1,200) # repeat keydown events every
    200ms

    def tick(self, dt):

        # process all keyboard & window events
        for event in pygame.event.get():
            cont = True
            if event.type == pygame.QUIT:
                cont = self.evm.keyboard_quit()
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_ESCAPE:
                cont = self.evm.keyboard_quit()
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_SPACE:
                cont = self.evm.keyboard_drop_bomb()
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_LEFT:
                cont =
    self.evm.keyboard_move_character(DIRECTION_LEFT)
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_RIGHT:
                cont
    =self.evm.keyboard_move_character(DIRECTION_RIGHT)
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_UP:
                cont =
    self.evm.keyboard_move_character(DIRECTION_UP)
            elif event.type == pygame.KEYDOWN and event.key ==
    pygame.K_DOWN:
                cont =
    self.evm.keyboard_move_character(DIRECTION_DOWN)
            # don't continue?
            if not cont: return False

        return True
```

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

from model import *
import pygame

#############################################################################
#                               VIEW                  #
#############################################################################

### Constants ###

FPS = 30
WIN_TITLE = "Bomber Man"
SPRITE_SIZE = 30 # 30x30 pixels
YELLOW = (255, 255, 0)
BLUE = (0, 0, 255)
RED = (255, 0, 0)

### Sprites ###

SPRITE_BACKGROUNDS = [ "images/misc/bg0.png",
    "images/misc/bg1.png", "images/misc/bg2.png" ]
SPRITE_BLANK = "images/misc/blank.png"
SPRITE_WALLS = [ "images/misc/wall0.png", "images/misc/wall1.png",
    "images/misc/wall2.png" ]
SPRITE_BOMB = "images/misc/bomb.png"
SPRITE_FIRE = "images/misc/fire.png"
SPRITE_FRUITS = [ "images/misc/banana.png",
    "images/misc/cherry.png" ]
SPRITE_DK = [ "images/dk/left.png", "images/dk/right.png",
    "images/dk/up.png", "images/dk/down.png" ]
SPRITE_ZELDA = [ "images/zelda/left.png",
    "images/zelda/right.png", "images/zelda/up.png",
    "images/zelda/down.png" ]
SPRITE_BATMAN = [ "images/batman/left.png",
    "images/batman/right.png", "images/batman/up.png",
    "images/batman/down.png" ]

### Class PyGameView ###

class GraphicView:

    # initialize PyGame graphic view
    def __init__(self, model, playername = ""):
        self.model = model
        self.width = model.map.width*SPRITE_SIZE
        self.height = model.map.height*SPRITE_SIZE
        # create window
        self.win = pygame.display.set_mode((self.width,
    self.height))
        # load sprites
        self.sprite_walls = [ pygame.image.load(sprite).convert()
    for sprite in SPRITE_WALLS ]
        self.sprite_backgrounds = [
    pygame.image.load(sprite).convert() for sprite in
    SPRITE_BACKGROUNDS ]
        self.sprite_blank =
    pygame.image.load(SPRITE_BLANK).convert()
        self.sprite_fruits = [
    pygame.image.load(sprite).convert_alpha() for sprite in
```

```python
         SPRITE_FRUITS ]
48            self.sprite_bomb =
         pygame.image.load(SPRITE_BOMB).convert_alpha()
49            self.sprite_fire =
         pygame.image.load(SPRITE_FIRE).convert_alpha()
50            sprite_dk = [ pygame.image.load(sprite).convert_alpha()
         for sprite in SPRITE_DK ]
51            sprite_zelda = [ pygame.image.load(sprite).convert_alpha()
         for sprite in SPRITE_ZELDA ]
52            sprite_batman = [
         pygame.image.load(sprite).convert_alpha() for sprite in
         SPRITE_BATMAN ]
53            self.sprite_characters = [sprite_dk, sprite_zelda,
         sprite_batman]
54            # init view
55            pygame.display.set_icon(self.sprite_bomb)
56            title = WIN_TITLE
57            if playername: title = WIN_TITLE + " (" + playername + ")"
58            pygame.display.set_caption(title)
59            self.font = pygame.font.SysFont('Consolas', 20)

60
61        # render map view
62        def render_map(self, m):
63            # win.blit(self.background, (0, 0))
64            for y in range(0, m.height):
65                for x in range(0, m.width):
66                    square = m.array[y][x]
67                    x0 = x*SPRITE_SIZE
68                    y0 = y*SPRITE_SIZE
69                    # walls
70                    if square == 'w':
71                        self.win.blit(self.sprite_walls[0], (x0, y0))
72                    elif square == 'x':
73                        self.win.blit(self.sprite_walls[1], (x0, y0))
74                    elif square == 'z':
75                        self.win.blit(self.sprite_walls[2], (x0, y0))
76                    # backgrounds
77                    elif square == '0':
78                        self.win.blit(self.sprite_backgrounds[0], (x0,
         y0))
79                    elif square == '1':
80                        self.win.blit(self.sprite_backgrounds[1], (x0,
         y0))
81                    elif square == '2':
82                        self.win.blit(self.sprite_backgrounds[2], (x0,
         y0))
83                    else:
84                        self.win.blit(self.sprite_blank, (x0, y0)) #
         blank

85
86        # render fruit view
87        def render_fruit(self, fruit):
88            x = fruit.pos[X] * SPRITE_SIZE
89            y = fruit.pos[Y] * SPRITE_SIZE
90            self.win.blit(self.sprite_fruits[fruit.kind], (x, y))

91
92        def render_bomb_explosion(self, bomb):
93            x0 = bomb.pos[X]
94            y0 = bomb.pos[Y]
95            for x in range(bomb.range[DIRECTION_LEFT],
         bomb.range[DIRECTION_RIGHT]+1):
96                self.win.blit(self.sprite_fire, (x*SPRITE_SIZE,
```

```python
        y0*SPRITE_SIZE))
97          for y in range(bomb.range[DIRECTION_UP],
        bomb.range[DIRECTION_DOWN]+1):
98              self.win.blit(self.sprite_fire, (x0*SPRITE_SIZE,
        y*SPRITE_SIZE))

99
100     def render_bomb_drop(self, bomb):
101         x = bomb.pos[X] * SPRITE_SIZE
102         y = bomb.pos[Y] * SPRITE_SIZE
103         self.win.blit(self.sprite_bomb, (x, y))
104         x0 = x + SPRITE_SIZE/2
105         y0 = y + SPRITE_SIZE/2
106         text = self.font.render(str(bomb.countdown), True, YELLOW)
107         rect = text.get_rect(center=(x0-5,y0+5))
108         self.win.blit(text, rect)

109
110     def render_bomb(self, bomb):
111         if(bomb.countdown == 0):
112             self.render_bomb_explosion(bomb)
113         elif(bomb.countdown > 0):
114             self.render_bomb_drop(bomb)

115
116     def render_character(self, character):
117         x = character.pos[X] * SPRITE_SIZE
118         y = character.pos[Y] * SPRITE_SIZE
119         sprite =
        self.sprite_characters[character.kind][character.direction]
120         self.win.blit(sprite, (x, y))

121
122     def render_player(self, player):
123         if not player: return
124         x = player.pos[X] * SPRITE_SIZE
125         y = player.pos[Y] * SPRITE_SIZE
126         pygame.draw.rect(self.win, RED, (x, y, SPRITE_SIZE,
        SPRITE_SIZE), 1)

127
128     # render PyGame graphic view at each clock tick
129     def tick(self, dt):
130         self.render_map(self.model.map)
131         for bomb in self.model.bombs:
132             self.render_bomb(bomb)
133         for fruit in self.model.fruits:
134             self.render_fruit(fruit)
135         for character in self.model.characters:
136             self.render_character(character)
137         self.render_player(self.model.player)
138         pygame.display.flip()
```