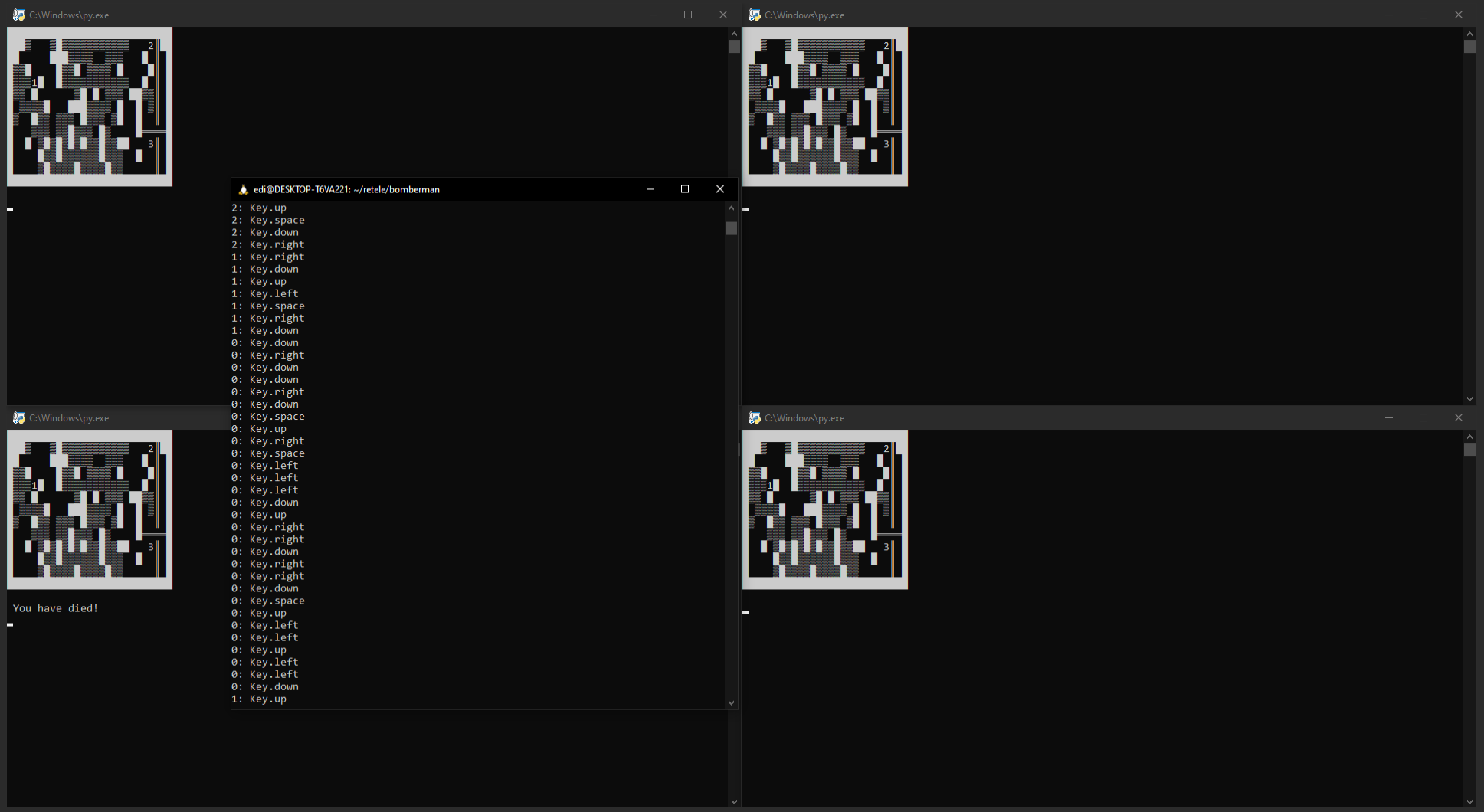
Bomberman

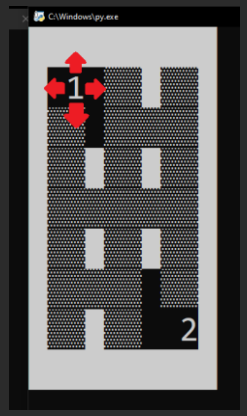
Tenche Eduard & Tuslea Andreea

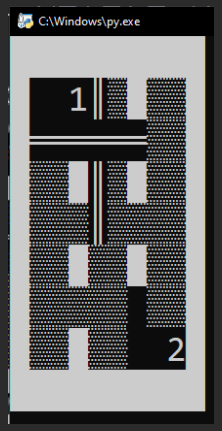


Jocul

Regulile jocului sunt simple:

- Un jucator se poate misca in 4 directii
- Jucatorul poate plasa o bomba oriunde se afla
- Bomba explodeaza dupa 3 secunde, explozia trecand prin toate spatiile libere, si distrugand maxim un perete distructibil in fiecare directie
- Peretii albi solizi sunt indistructibili
- Orice jucator aflat in explozia unei bombe este eliminat
- Scopul jocului e sa ii elimini pe ceilalti jucatori





Implementare

Am folosit Python pentru ca functioneaza independent de sistemul de operare

Clientul, totusi, functioneaza doar pe Windows din cauza ca nu am gasit metode usoare de a captura input-ul utilizatorului in timp real care sa functioneze si pe alte platforme.

El doar se conecteaza, captureaza input-uri pe care le trimite la server, si deseneaza harta de fiecare data cand primeste o actualizare de la acesta.

Server-ul poate rula si pe Linux, si are grija de toate regulile jocului.

Inainte de a porni server-ul propriu-zis, se alege o harta, care este sub format de .bombermap si contine doar simboluri ASCII care reprezinta elementele jocului:

```
1  XXXXXXXXXX
2  X1  OOXOOX
3  XOO OOOOOX
4  XOOXOOXOOX
5  XOOOOOOOOX
6  XOOXOOXOOX
7  XOOOOO OOX
8  XOOXOO  2X
9  XXXXXXXXXX
```

Un exemplu de fisier .bombermap care contine o harta

- 'X' reprezinta peretii indestructibili
- 'O' ii reprezinta pe cei destructibili
- Orice cifra reprezinta start-ul unui jucator

Apoi server-ul asteapta pana toate starturile sunt umplute, deci numarul de jucatori depinde de harta.

Pentru fiecare client este creat un nou thread, conexiunea acestuia fiind salvata intr-o lista. Server-ul apoi interpreteaza input-urile primite de la fiecare, si actualizeaza matricea interna, pe care o trimite apoi la client pentru a fi printata pe ecran.

Cand ramane un singur jucator, jocul se termina.

Scurta demonstratie video:
youtu.be/BOvnVo4gnk8

Cod

server

```
import socket, time
from _thread import *

#### Metodele folosite in program

# Inlocuieste un caracter din harta cu altul (doar un shortcut)
def replaceCharAtIndex(i, j, char):
    map[i] = map[i][:j] + char + map[i][j + 1:]


# Loop-ul care primeste input de la client, creaza un nou thread pentru fiecare client si ii salveaza intr-o variabila
globala
def on_new_client(id, connection, address):
    global connections
    connection.sendall(str("Connected as Player" + str(id+1)).encode('utf-8'))

    connections.append((connection, id))
    while not ended:
        data = connection.recv(1024)
        data = data.decode('utf-8')
        if data:
            print(str(id) + ": " + str(data))
```

```
            handleInput(id, data)


# Functie care verifica input-ul
def handleInput(id, key):
    global gameStateChanged

    position = None
    index = 0
    for tuple in playerPos: # Gasim jucatorul cu id-ul cerut
        if tuple[0] == id:
            position = (tuple[1], tuple[2])
            break
        index+=1

    checkedPosition = None
    placeBomb = False
    match key: # Verifica tasta apasata
        case "Key.up":
            checkedPosition=(position[0]-1, position[1])
        case "Key.down":
            checkedPosition=(position[0]+1, position[1])
        case "Key.right":
            checkedPosition = (position[0], position[1]+1)
        case "Key.left":
            checkedPosition = (position[0], position[1]-1)
        case "Key.space":
            placeBomb = True
        case _:
            pass

    # Daca e una dintre sageti si directia nu este blocata/inafara hartii
    if checkedPosition and checkedPosition[0] >= 0 and checkedPosition[0] < len(map) and checkedPosition[1] >= 0 and
checkedPosition[1] < len(map[0]):
        if map[checkedPosition[0]][checkedPosition[1]] != 'O' and map[checkedPosition[0]][checkedPosition[1]] != 'X':
            playerPos[index] = (id, checkedPosition[0], checkedPosition[1])
    elif placeBomb and map[position[0]][position[1]] != 'Z': # Sau pune o bomba
        replaceCharAtIndex(position[0], position[1], 'Z')
        bombTimeStamps.append((position[0], position[1], time.time())) # Adauga bomba in lista

    gameStateChanged = True # Harta trebuie redesenata


# Functie care explodeaza bomba
def blowUpBomb(timeStamp):
    global gameStateChanged
```

```
    #Caracterele particulelor de explozie sunt - si | pentru orizontal si vertical respectiv
    replaceCharAtIndex(timeStamp[0], timeStamp[1], '-')
    explosionTimeStamps.append((timeStamp[0], timeStamp[1], time.time()))

    createExplosion(timeStamp[0], timeStamp[1], -1, 0, '|') #Creaza o explozie in sus
    createExplosion(timeStamp[0], timeStamp[1], 1, 0, '|')  # In jos
    createExplosion(timeStamp[0], timeStamp[1], 0, -1, '-') # In stanga
    createExplosion(timeStamp[0], timeStamp[1], 0, 1, '-') #In dreapta

    gameStateChanged = True # Harta trebuie redesenata


# Functie care creaza particulele de explozie in directia alesa
def createExplosion(i, j, diffI, diffJ, char):

    # (Avertizare: Urmeaza cod oribil)

    target = (i, j)
    while map[i][j] != 'X': # Merge pana la un perete indestructibil
        target = (target[0] + diffI, target[1] + diffJ)

        # Verifica daca explozia loveste un jucator
        for tuple in playerPos:
            if tuple[1] == target[0] and tuple[2] == target[1]:
                kill(tuple[0]) # Il omoara daca da


        # Break daca loveste un perete indestructibil
        if map[target[0]][target[1]] == 'X':
            break

        # Daca este un perete destructibil, il distruge si da break abia dupa
        shouldBreak = False
        if map[target[0]][target[1]] == 'O':
            shouldBreak = True

        # Inlocuieste toate spatiile afectate cu caracterul ales
        replaceCharAtIndex(target[0], target[1], char)

        # Adauga particulele in lista lor
        explosionTimeStamps.append((target[0], target[1], time.time()))

        if shouldBreak:
            break
```

```
# Functie care omoara un jucator
def kill(id):
    global playerPos
    for tuple in playerPos:
        if tuple[0] == id:
            playerPos.remove(tuple)

    print("Player " + str(id) + " has died.")

# Deschide fisierul cu harta
#
# Legenda fisier harta:
#
# 'X' = perete indestructibil
# 'O' = perete destructibil
# [0-9] = spawn point pentru jucatori
# ' ' = spatiu liber
#
#
filename = input("Type Map Name: ")
file = open(filename + ".bombermap")


# Variabilele globale ale jocului

ended = False # True cand jocul s-a terminat

map = [] # Matrice cu String-uri care contine harta. Metoda destul de stupida de implementare.
playerPos = [] # Array cu Tuples care contin informatii despre pozitia jucatorilor
connections = [] # Array cu toti clientii conectati
gameStateChanged = False # True cand trebuie desenata din nou harta de catre clienti
bombTimeStamps = [] # Contine toate bombele si cand au fost create
explosionTimeStamps = [] # Contine toate particulele de explozie si cand au fost create

# Citirea fisierului care contine harta
lineNr = 0
for line in file:
    charNr = 0
    for char in line:
        if char.isdigit():
            playerPos.append((int(char)-1, lineNr, charNr)) # Salveaza pozitiile jucatorilor, marcati cu numere
        charNr+=1
    map.append(line.replace("\n", "")) # Eliminare \n
    lineNr+=1

for player in playerPos:
```

```
    replaceCharAtIndex(player[1],player[2], ' ') # Jucatorii nu sunt continuti pe harta, ci in playerPos, deci sunt
eliminati

# Crearea server-ului
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("127.0.0.1", 8008))
s.listen()


# Asteapta sa se conecteze destui jucatori sa umple toate spawn point-urile de pe harta aleasa
id = 0
while id < len(playerPos):
    connection, address = s.accept()

    # Deschide un nou thread cu propriul loop
    start_new_thread(on_new_client, (id, connection, address))
    id+=1

# Deseneaza harta pentru clienti cand s-au conectat toti jucatorii
gameStateChanged = True

# Loop-ul jocului
while True:


    if ended == False:
        # Un timestamp curent
        currentTime = time.time()

        # Verifica daca o bomba trebuie sa explodeze (daca timestamp-ul crearii bombei este acum 3+ secunde)
        for timeStamp in bombTimeStamps:
            if currentTime - timeStamp[2] > 3.0:
                blowUpBomb(timeStamp)
                bombTimeStamps.remove(timeStamp)

        # Verifica daca exista un singur jucator (conditie de castig)
        if len(playerPos) == 1:
            gameStateChanged = False
            ended = True
            for conn in connections:
                conn[0].sendall(str("Victory for Player " + str(playerPos[0][0] + 1) + "!").encode('utf-8'))

        # Verifica daca o particula trebuie sa dispara

        for timeStamp in explosionTimeStamps:
            if currentTime - timeStamp[2] > 0.5:
```

```
        replaceCharAtIndex(timeStamp[0], timeStamp[1], ' ')
        explosionTimeStamps.remove(timeStamp)


# Daca gamestate-ul s-a schimbat (orice schimbare pe harta)
if gameStateChanged:
    gameStateChanged = False

    # Deseneaza harta pentru fiecare jucator
    for conn in connections:
        output = ""

        for i in range(len(map)):
            for j in range(len(map[0])):
                isPlayerPos = False
                # Verifica daca caracterul curent este pozitia unui jucator
                for tuple in playerPos:
                    if i == tuple[1] and j == tuple[2]:
                        # Deseneaza-l peste daca da
                        output+=str(tuple[0]+1)
                        isPlayerPos = True
                        break

                if not isPlayerPos:
                    output+=map[i][j]

            output+='\n'
        #for x in range(len(map)*2+1): # spaghetti code
        #    output+='\n'

        # Metoda stupida de a inlocui caracterele ASCII cu lucruri mai fancy
        output = output.replace('O', '▒')
        output = output.replace('X', '▉')
        output = output.replace('|', '╫')
        output = output.replace('-', '=')
        output = output.replace('Z', 'O')

        foundPlayer = False
        for player in playerPos:
            if player[0] == conn[1]:
                foundPlayer = True

        if not foundPlayer:
            output+="\n You have died!"

        conn[0].sendall(output.encode('utf-8'))
```

Cod client

```
# Intentionat sa fie rulat pe windows, deoarece nu am gasit metode simple de a captura input in realtime in Unix

import socket, win32gui, win32process, os
from pynput import keyboard

# Creare client
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#Conectare la IP
###IP = input("IP: ")
IP = "localhost"
s.connect((IP, 8008))

# O functie care trimite input-ul de la tastatura serverului
def on_press(key):
    focus_window_pid = win32process.GetWindowThreadProcessId(win32gui.GetForegroundWindow())[1]
```

```
    current_process_pid = os.getppid()
    if focus_window_pid == current_process_pid:
        s.sendall(str(key).encode('utf-8'))

# Asculta pentru input-uri
listener = keyboard.Listener(
        on_press=on_press,
        on_release=None)
listener.start()


print("Successfully connected to " + IP)


# Loop pentru desenare imediata a informatiilor primite de la server
while True:
    data = s.recv(1024)

    if data:
        os.system('cls')
        print(data.decode('utf-8'))
```