MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
CATEDRA CALCULATOARE

Inteligență Artificială

Nume:
Blaj Sergiu-Emanuel
Borbei Raul-Aurelian

Grupa: 30235

Email:
sergiumr@yahoo.com
borbeiraul@yahoo.com

Profesor îndrumator: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Cuprins

Tema 1

A1: Search - Pacman Lucky blocks

1.1 Introducere

Scopul lucrării

Scopul temei era să ne familiarizăm cu conceptele de bază din Python prin intermediul proiectului Pacman Berkeley. Astfel proiectul a conținut două părți: o primă parte în care am invățat concepte de search în Python aplicate agentului Pacman și o a doua, la alegere prin care noi a trebuit să aducem ceva în plus jocului Pacman.

Alegerea facută de noi pentru partea în plus a proiectului adaugă jocului Pacman un nou mod de joc care cu siguranță aduce pentru jucator dorința de a juca. Cu toate acestea, natura jocului ales nu a permis să fie utilizați algoritmii de search scriși în cadrul orelor de laborator.

1.2 Keyboard Agent

1.2.1 Jocul

Vechi

Jocul în sine a rămas în mare parte acelasi: Pacman este controlat de către jucător pe o tablă de joc 2D. Tabla este formată dintr-un labirint de ziduri între care există mâncare(punctele mici)și lucky blocks(punctele mari). Punctele mici cresc scorul jucătorului și prin mâncarea tuturor punctelor de pe tabla de joc, jucătorul termină jocul. De asemenea pe tablă se află și fantome care dacă îl ating pe Pacman acesta pierde jocul instant.

Nou

Elementul de noutate din joc îl constituie lucky blocks. Punctele mari care înainte îi permiteau lui Pacman să mănânce fantomele pe o durata de 30 de secunde acum fac mai multe acțiuni. Când Pacman mănâncă un lucky block se alege random unul dintre următoarele lucruri: invincibilitate, viteză, culoare, dimensiune, apariția lui Ms.Pacman undeva pe tabla de joc, teleportare, câștig imediat. Însă, toate aceste lucruri au și opusele în joc și astfel Pacman poate să își piardă din viteză, sau să se micșoreze și în loc de câștig imediat poate fi un game over.

1.2.2 Logica din spate

În scopul creeri noului joc a fost necesar să modificam mai multe file din proiectul de multiagent de la berkeley.

În game.py au fost adăugate câteva variabile care să gestioneze efectele mâncării, vieților lui Pacman și ale timerului pentru efectul de mâncare.

În graphicsDisplay.py am modificat culoarea mâncării, durata de timp de la mâncare, am introdus o funcție care să aleagă random efectul de la mâncare, o clasă noua care conține câteva variabile necesare, am modificat panoul de scor să se afișeze și efectul mâncării, am introdus o noua funcție pentru vieți și există mai multe modificări care se ocupă de efectele mâncării în joc cum ar fi: dimensionarea lui Pacman, viteza sau culoarea.

În pacman.py sunt funcțiile care se ocupă de efectele mâncării cu tot cu partea de apel a funcțiilor. A fost necesar să schimbam puțin și funcțiile de la agentul ghost cu scopul de a o puta adauga pe Ms.Pacman.

1.2.3 Rulare

Pentru rularea jocului, jucătorul trebuie să introducă urmatoarea linie în terminalul de comandă deschis din proiect:

python pacman.py -p KeyboardAgent -l madeMaze -k 2

1.3 Mod de joc

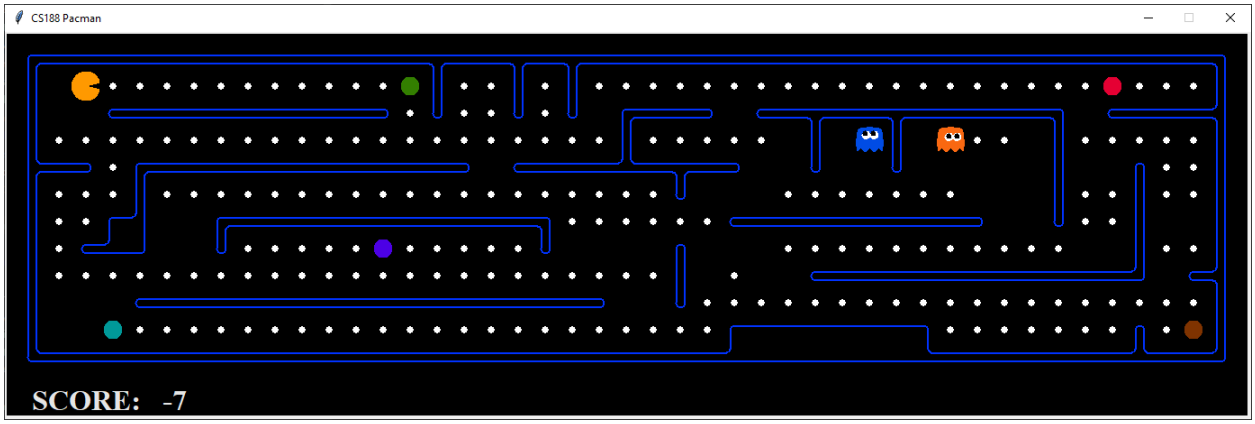
Jucătorul îl controlează pe Pacman cu ajutorul săgeților de la tastatură. Jocul poate fi câștigat acum în mai multe moduri. Primul mod este modul clasic de câștig după ce Pacman a mâncat toată mâncarea de pe mapă. Pacman poate de asemneea să câștige jocul prin mâncarea unui lucky block de instant win, în acest caz se va afișa un mesaj corespunzător, se va închide mapa și va fi calculat scorul obținut, acesta apărând în consolă. Un al treilea mod de a câștiga jocul este tot prin intermediul unui lucky block, în cazul în care acesta o va genera pe Ms.Pacman. Dupa generarea ei jucătorul ar trebui să se axseze să o găsească deoarece la întâlnirea dintre cei doi jocul se încheie cu un câștig garantat.

1.4 Bad luck

Cum este și normal jocul poate fi și pierdut și datorită faptului că există mai multe metode de câștig este doar normal să existe mai multe metode de a pierde. Modul clasic este atunci când Pacman este atins de o fantoma: mapa se închide și pe consolă se afișează scorul și mesajul de pierdere. Alte moduri de a pierde sunt asistate de câteva lucky blocks care nu sunt chiar așa de lucky... Pacman poate primi după mâncarea unui lucky block un freze de 30 de secunde, timp în care o fantomă îl poate atinge ușor sau poate primi o încetinire la viteză, astfel fantomele îl vor prinde mai ușor. Există de asemenea un super unlucky block care după consum termină jocul cu o pierdere. Orice alte bug-uri, cunoscute sau nu, apărute în urma modificării jocului original, care duc la închiderea mapei de joc și afișarea unui mesaj de eroare sunt considerate de asemenea noi modalități de pierdere.

1.5 Mapă nouă

Pentru testare și rularea jocului am creat un custom map madeMaze.lay pe care l-am inclus în folderul layouts a jocului. Jocul însă poate fi jucat pe orice mapă existentă sau pe care utilizatorul o crează, însă trebuie specificată la rulare.



Tema 2

A2: Logics - Minefield game

2.1 Introducere

Scopul lucrării

În viața de zi cu zi luăm decizii bazate pe raționamente logice fără să fim conștienți de întregul proces care conduce la rezultat. În termeni academici, întregul proces de gândire poate fi transpus sub forma logicii propoziționale, ceea ce ne oferă capacitatea de a înțelege fiecare etapă a raționamentului.

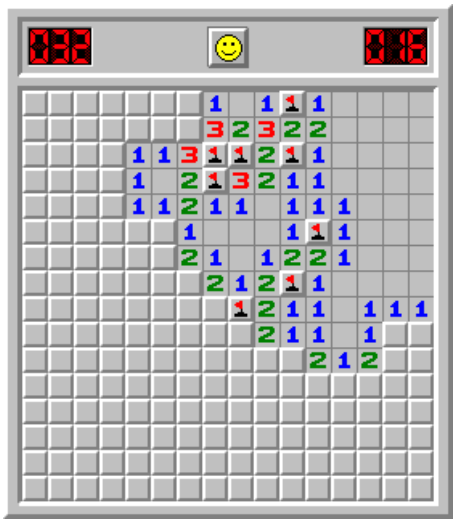
Scopul temei este familiarizarea studenților cu conceptele de first order logic prin intermediul unui joc interactiv. Conceptele logice pot fi transpuse în lumea virtuală prin intermediul unui puzzle de tip grid. Agentul uman va fi responsabil de deciziile luate în lumea jocului, iar fiecare decizie va avea consecințe negative sau pozitive.

2.2 Keyboard Agent

2.2.1 Jocul

Background

Cu siguranță toți suntem familiari cu celebrul joc Minesweeper în care ni se pune în față o tablă de joc formată din celule organizate pe rânduri și coloane pe care apăsând puteam să afișăm ce se află în spate. Regula jocului spune că prima apăsare este întotdeauna o celulă safe după care începe partea de logică. Fiecare celulă pe care am afișat-o deja conține în ea un număr între 1 și 8 (0 fiind de fapt reprezentat de o celulă goală), număr care semnifica câte celule vecine conțin o mină. Folosind logica, jucătorul poate să își dea seama care celule sunt safe și care conțin o mină. Scopul jocului este să descoperi toate celulele cu mine, sau altfel spus, să descoperi toate celulele safe.



Jocul a fost readaptat cu denumirea de MineFOL diferența fiind că în loc ca celulele să ne spună câte mine vecine au, unele dintre celule conțin mesaje în FOL care ne dau indicii fie despre locația minelor pe tabla de joc, fie despre care celule sunt safe.

Implementare

Folosind o interfață grafică creată în Python jucătorului i se prezintă în față o tablă de joc de 8x8 pe care își poate muta caracterul folosind săgețile de la tastatură. Tabla de joc conține un număr necunoscut de mine plasate random în celule, celulele care nu conțin mine fiind considerate celule safe. Primul indiciu despre joc se află întotdeauna în celula (1,1) de unde și pornește caracterul, astfel ea fiind mereu o celula safe. Dacă jucatorul descifrează indiciul și își mută caracterul în conformitate, el va primi puncte pentru trecerea printr-o celulă pe care Prover9 o consideră ca fiind safe. În schimb dacă mută caracterul într-o celulă despre care nu se știe că este safe și ea nu conține o mină acestuia i se vor scădea puncte. În cazul în care caracterul ajunge într-o celulă cu mina, jocul se termină cu mesaj de "Game over".

2.2.2 Logica din spate

Jocul este creat folosind limbajul Python și pentru validarea celulelor safe se apelează în spate comenzi de prover9 la fiecare pas făcut de către jucător.

Fișierul minefield.py conține funcțiile necesare pentru crearea, afișarea și gestionarea interfeței grafice. Tot în cadrul acestui fișier avem o funcție care apelează comanda de prover9 pentru a valida sau nu celula ce urmeaza sa fie accesată.

Mapele de joc și mesajele în FOL sunt stocate separat în format .txt și sunt încărcate automat la rulare folosind map_path.

2.2.3 Rulare

Pentru rularea jocului următorii pași sunt necesari:

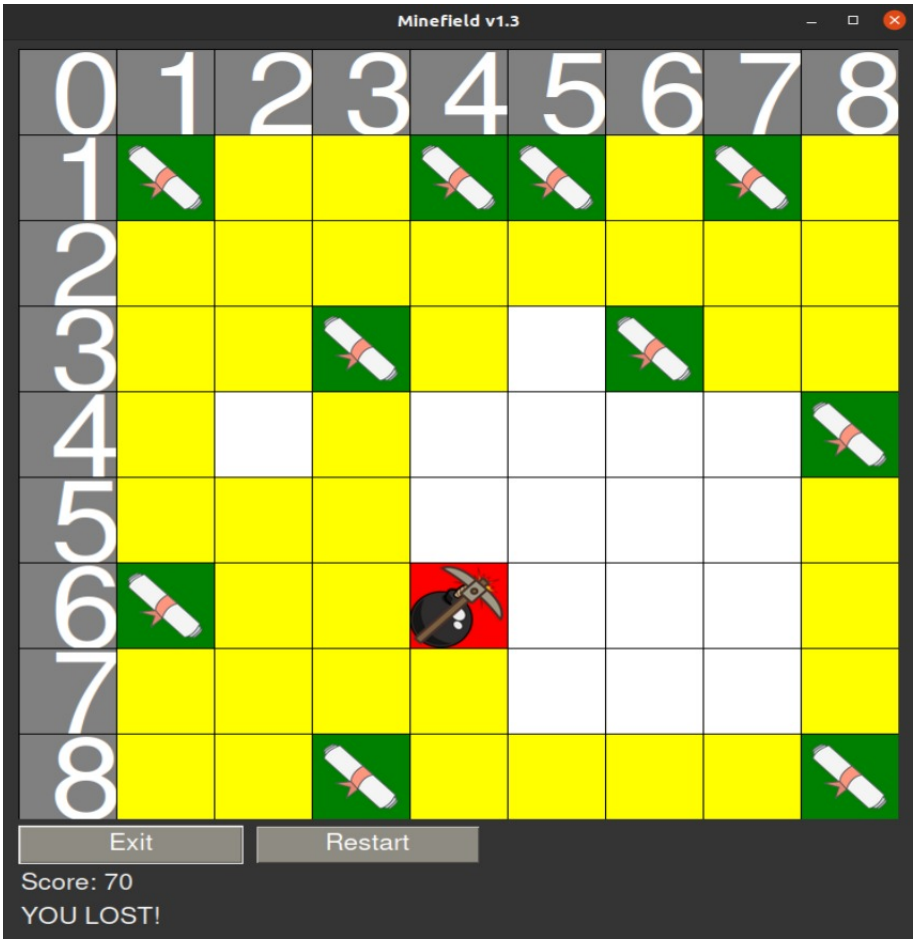
1. clone/ download the project
2. open folder with vscode/ pycharm
3. python -m venv venv

4. .\venv\Scripts\activate
5. pip install PySimpleGUI
6. pip install numpy
7. pip install pillow
8. python minefield.py

notă: prover9 trebuie să fie instalat pe sistem

2.3 Mod de joc

Jocul poate fi jucat folosind tastele up, down, left și right de la tastatură. Jucătorul pornește mereu din celula (1, 1) care este situată în partea din stânga-sus a tablei de joc și conține mereu un mesaj cu indicii. Jucătorul trebuie să citească și să înțeleagă indiciul care îi va da informații despre următoarele mișcări posibile. Dacă a descifrat bine indiciul și merge în partea corectă va primi puncte, dacă în schimb merge într-o celulă de care înca nu se știe că este safe i se vor scădea puncte. Jocul se termină când jucătorul a trecut prin toate celulele safe sau în momentul în care trece printr-o celulă care conține o mină.



2.3.1 Mapa

Jocul a fost implementat pe mapa de 8x8, pătratul de start fiind mereu cel din stânga sus, numerotat cu (1, 1). Am implementat pentru joc 3 mape cu dificultate tot mai mare datorată mesajelor și numărului de mine de pe mapă. Prima mapă are 4 bombe, a doua 5, iar a treia are 6.

De asemenea jucătorul poate să își implementeze propriile mape folosind următoarele reguli:

0 1 2 3 - axa x
1
2
3 - axa y

notă: axele sunt inversate

conținutul mapei ar trebui să fie:
% - indicele de axă
M - mesaj
B - mină
. - celulă goală
= - separă harta de mesaje
i j mesaj_text

De exemplu:

```
%%%%%
%M.M.
%....
%.B..
%...%
=====
1 1 exemplu de mesaj
3 1 mesajul cu numarul 2
```

Tema 3

A3: Planning

Appendix a

Cod original - tema 1

```
# game.py

self.luckyFoodTimer = 0
self.luckyFood = ""
self.luckyFoodColor = "WHITE"
self.lives = 1
self.ghostsEaten = 0
self.draw = True

#############...#############

state.luckyFoodTimer = self.luckyFoodTimer
state.luckyFood = self.luckyFood
state.luckyFoodColor = self.luckyFoodColor
state.lives = self.lives
state.draw = self.draw
state.ghostsEaten = self.ghostsEaten

#graphicsDisplay.py

CAPSULE_COLORS = [
    formatColor(0.0, 0.6, 0.6),
    formatColor(0.5, 0.2, 0.0),
    formatColor(0.3, 0.0, 0.9),
    formatColor(0.2, 0.5, 0.0),
    formatColor(0.9, 0.0, 0.2),
]
foodTexts = ["Pacman freeze", "Pacman speed decrease", "Ghost freeze", "
    Ghost speed decrease", "Pacman size decrease", "Pacman size increase", "
    Immunity"]

class UsefulVariables:
    PACMAN_COLOR = formatColor(1.0, 0.6, 0.0)
    PACMAN_SPEED = 1.0
    PACMAN_SIZE_REQUEST = False
    PACMAN_SIZE = 0.5
    PACMAN_TELEPORT = False
    PACMAN_COORDINATES = (-1, -1)

    MS_PACMAN_ID = -1
    MS_PACMAN = None

    GHOST_SIZE = 0.65
    GHOST_SPEED = 1.0
    NO_GHOSTS = False
```

```
44
45      COLLISION_TOLERANCE = 0.7
46
47  Variables = UsefulVariables
48
49
50  def drawPane(self):
51      self.scoreText = text(self.toScreen(0, 0), self.textColor, "SCORE:    0"
        , "Times", self.fontSize, "bold")
52      self.scoreText = text(self.toScreen(0, 0), self.textColor, "SCORE:    0"
        , "Times", self.fontSize, "bold")
53      self.foodText = text(self.toScreen(250, 0), self.textColor, "", "Times",
         self.fontSize, "bold")
54      self.livesText = text(self.toScreen(0, 30), self.textColor, "LIVES: ", "
        Times", self.fontSize, "bold")
55      self.ghostsEaten = text(self.toScreen(250, 30), self.textColor, f"GHOSTS
         EATEN: 0 x {SYMBOL_GHOST}", "Times",
56                              self.fontSize, "bold")
57
58  def updateLives(self, lives):
59      changeText(self.livesText, f"LIVES: {lives} x {SYMBOL_HEART}")
60
61  def updateFood(self, food, time, color):
62      changeColor(self.foodText, color)
63
64      timeText = f"({time})" if food in foodTexts else ""
65
66      if time == 0:
67          changeText(self.foodText, "")
68      else:
69          changeText(self.foodText, f"FOOD: {food} {timeText}")
70
71  def updateGhostsEaten(self, ghostsEaten):
72      changeText(self.ghostsEaten, f"GHOSTS EATEN: {ghostsEaten} x {
        SYMBOL_GHOST}")
73
74  ############...############
75
76  if agentState.isPacman:
77              self.infoPane.updateGhostsEaten(agentState.ghostsEaten)
78              self.infoPane.updateLives(agentState.lives)
79              self.infoPane.updateFood(agentState.luckyFood, agentState.
        luckyFoodTimer, agentState.luckyFoodColor)
80
81   def getGhostColor(self, ghost, ghostIndex):
82          if ghost.scaredTimer > 0:
83          if Variables.NO_GHOSTS == True:
84              return BACKGROUND_COLOR
85          elif Variables.MS_PACMAN_ID == ghostIndex:
86              return formatColor(0.7, 0.1, 0.7)
87          elif ghost.scaredTimer > 0:
88              return SCARED_COLOR
89          else:
90              return GHOST_COLORS[ghostIndex]
91
92   def moveGhost(self, ghost, ghostIndex, prevGhost, ghostImageParts):
93          if ghostIndex == Variables.MS_PACMAN_ID:
94              for ghostImagePart in ghostImageParts:
95                  move_by(ghostImagePart, (-1000, -1000))
96              refresh()
```

```
97
98              if Variables.MS_PACMAN != None:
99                  moveCircle(Variables.MS_PACMAN, (99999, 99999), 0)
100             Variables.MS_PACMAN = PacmanGraphics.drawPacman(self, ghost, 0,
    True)
101
102         if Variables.NO_GHOSTS == True:
103             for ghostImagePart in ghostImageParts:
104                 move_by(ghostImagePart, (-1000, -1000))
105             refresh()
106             return
107
108 #layouts/madeMaze.lay
109
110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
111 % P...........o%..%.%...................o...%
112 %  %%%%%%%%%%.%..%.% %%%% %%%%%%%%%%%% %%%%%
113 %.....................%..... %GG% ... %.....%
114 %%%.%%%%%%%%%%%%%%% %%%%%%%%%  %  %     %  %..%
115 %...%.................%    .......    %..%..%
116 %..%%  %%%%%%%%%%%%%%.....%%%%%%%%%%  %..%  %
117 %.%%%  %.....o.....%    %   ...........  %..%
118 %.....................% .  %%%%%%%%%%%% %%
119 %   %%%%%%%%%%%%%%%%%  %.................%
120 %  o.....................%%%%%%%%.......%.o%
121 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
122
123 #pacman.py
124
125 def handleLuckyFood(state, pacman):
126     pacman.luckyFoodTimer = LUCKYFOOD_TIME
127
128     luckyFoodId = random.randint(0, 14)
129
130     if luckyFoodId == 0:
131         PacmanRules.handlePacmanFreeze(pacman)
132     elif luckyFoodId == 1:
133         PacmanRules.handlePacmanSpeedDecrease(pacman)
134     elif luckyFoodId == 2:
135         PacmanRules.handleGhostFreeze(state, pacman)
136     elif luckyFoodId == 3:
137         PacmanRules.handleGhostSpeed(state, pacman)
138     elif luckyFoodId == 4:
139         PacmanRules.handlePacmanSizeMinus(pacman)
140     elif luckyFoodId == 5:
141         PacmanRules.handlePacmanSizePlus(pacman)
142     elif luckyFoodId == 6:
143         PacmanRules.handlePacmanColor(pacman)
144     elif luckyFoodId == 7:
145         PacmanRules.handleImmunity(state, pacman)
146     elif luckyFoodId == 8:
147         PacmanRules.handleNoGhosts(state, pacman)
148     elif luckyFoodId == 9:
149         PacmanRules.handleInstantWin(state, pacman)
150     elif luckyFoodId == 10:
151         PacmanRules.handleInstantLose(state, pacman)
152     elif luckyFoodId == 11:
153         PacmanRules.handleTeleport(state, pacman)
154     elif luckyFoodId == 12:
155         PacmanRules.handlePacmanLives(state, pacman)
```

```
    elif luckyFoodId == 13:
        PacmanRules.handlePacmanScore(state, pacman)
    else:
        PacmanRules.handleMsPacman(state, pacman)


handleLuckyFood = staticmethod(handleLuckyFood)


def getColor(color):
    if color == "WHITE":
        return formatColor(1.0, 1.0, 1.0)
    if color == "GREEN":
        return formatColor(0.0, 1.0, 0.0)
    if color == "RED":
        return formatColor(1.0, 0.0, 0.0)
    if color == "PINK":
        return formatColor(0.7, 0.1, 0.7)


getColor = staticmethod(getColor)


def handlePacmanFreeze(pacman):
    pacman.luckyFood = "Pacman freeze"
    Variables.PACMAN_SPEED = 0.0
    pacman.luckyFoodColor = PacmanRules.getColor("RED")


def handlePacmanSpeedDecrease(pacman):
    pacman.luckyFood = "Pacman speed decrease"
    Variables.PACMAN_SPEED = 0.5
    pacman.luckyFoodColor = PacmanRules.getColor("RED")


def handleGhostFreeze(state, pacman):
    if Variables.NO_GHOSTS:
        return PacmanRules.handleInstantWin(state, pacman)

    pacman.luckyFood = "Ghost freeze"
    Variables.GHOST_SPEED = 0.0
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handleGhostSpeed(state, pacman):
    if Variables.NO_GHOSTS:
        return PacmanRules.handleInstantWin(state, pacman)

    pacman.luckyFood = "Ghost speed decrease"
    Variables.GHOST_SPEED = 0.5
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handlePacmanSizeMinus(pacman):
    pacman.luckyFood = "Pacman size decrease"
    Variables.PACMAN_SIZE = 0.3
    pacman.luckyFoodColor = PacmanRules.getColor("RED")


def handlePacmanSizePlus(pacman):
```

```
    pacman.luckyFood = "Pacman size increase"
    Variables.PACMAN_SIZE = 0.7
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handlePacmanColor(pacman):
    pacman.luckyFood = "Random color"
    Variables.PACMAN_COLOR = formatColor(random.random(), random.random(),
   random.random())
    pacman.luckyFoodColor = PacmanRules.getColor("WHITE")


def handleImmunity(state, pacman):
    if Variables.NO_GHOSTS:
        return PacmanRules.handleInstantWin(state, pacman)

    pacman.luckyFood = "Immunity"
    for index in range(1, len(state.data.agentStates)):
        state.data.agentStates[index].scaredTimer = LUCKYFOOD_TIME
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handleNoGhosts(state, pacman):
    if Variables.NO_GHOSTS:
        return PacmanRules.handleInstantWin(state, pacman)

    pacman.luckyFood = "No ghosts"
    Variables.GHOST_SIZE = 0
    Variables.NO_GHOSTS = True
    Variables.COLLISION_TOLERANCE = -1
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handleInstantWin(state, pacman):
    pacman.luckyFood = "Instant win"
    state.data._win = True
    state.data.score += 1000
    pacman.luckyFoodColor = PacmanRules.getColor("GREEN")


def handleInstantLose(state, pacman):
    pacman.luckyFood = "Instant lose"
    state.data._lose = True
    state.data.score -= 1000
    pacman.luckyFoodColor = PacmanRules.getColor("RED")


def handleTeleport(state, pacman):
    pacman.luckyFood = "Teleport"

    Variables.PACMAN_TELEPORT = True

    x, y = 0, 0
    while state.hasWall(x, y):
        x, y = random.randint(1, state.data.layout.walls.width - 1), random.
   randint(1,

            state.data.layout.walls.height - 1)

    Variables.PACMAN_COORDINATES = x, y
```

```
273
274     pacman.luckyFoodColor = PacmanRules.getColor("WHITE")
275
276
277 def handlePacmanLives(state, pacman):
278     pacman.luckyFood = "Life point added"
279     pacman.lives += 1
280     pacman.luckyFoodColor = PacmanRules.getColor("GREEN")
281
282
283 def handlePacmanScore(state, pacman):
284     randomScore = random.randint(-50, 100)
285
286     if randomScore > 0:
287         pacman.luckyFood = f"Score modified (+{randomScore})"
288         pacman.luckyFoodColor = PacmanRules.getColor("GREEN")
289     else:
290         pacman.luckyFood = f"Score modified ({randomScore})"
291         pacman.luckyFoodColor = PacmanRules.getColor("RED")
292
293     state.data.score += randomScore
294
295
296 def handleMsPacman(state, pacman):
297     if Variables.NO_GHOSTS or len(state.data.agentStates) == 1:
298         return PacmanRules.handleInstantWin(state, pacman)
299
300     pacman.luckyFood = "Ms Pacman"
301
302     Variables.MS_PACMAN_ID = random.randrange(1, len(state.data.agentStates)
    )
303
304     pacman.luckyFoodColor = PacmanRules.getColor("PINK")
305
306
307 def revertChanges():
308     Variables.PACMAN_SPEED = 1.0
309     Variables.PACMAN_COLOR = formatColor(1.0, 0.6, 0.0)
310     Variables.PACMAN_SIZE = 0.5
311     Variables.PACMAN_TELEPORT = False
312     Variables.PACMAN_COORDINATES = (-1, -1)
313
314     Variables.GHOST_SPEED = 1.0
315     Variables.GHOST_SIZE = 0.5
316
317
318 revertChanges = staticmethod(revertChanges)
319
320
321 def decrementFoodTimer(pacmanState):
322     timer = pacmanState.luckyFoodTimer
323     pacmanState.luckyFoodTimer = max(0, timer - 1)
324     if timer == 0:
325         PacmanRules.revertChanges()
326
327
328 decrementFoodTimer = staticmethod(decrementFoodTimer)
329
330
331 def collide(state, ghostState, agentIndex):
```

```
    if ghostState.scaredTimer > 0:
        pacman = state.data.agentStates[0]

    if agentIndex == Variables.MS_PACMAN_ID:
        state.data._win = True
    elif ghostState.scaredTimer > 0:
        state.data.scoreChange += 200
        GhostRules.placeGhost(state, ghostState)
        ghostState.scaredTimer = 0
        pacman.ghostsEaten += 1
        # Added for first-person
        state.data._eaten[agentIndex] = True
    else:
        if not state.data._win:
            state.data.scoreChange -= 500
            state.data._lose = True
    elif pacman.lives > 1:
    state.data.scoreChange -= 200
    GhostRules.placeGhost(state, ghostState)
    pacman.lives -= 1

elif not state.data._win:
state.data.scoreChange -= 500
state.data._lose = True
collide = staticmethod(collide)
```

Appendix b

Cod original - tema 2

```
'''
Name: minefield.py
Authors: Blaj Sergiu, Borbei Raul
Description: minefield game
'''

import os
import numpy
import PySimpleGUI
from tkinter import *
from PIL import ImageTk, Image

MAPS_FOLDER = './resources/maps'
IMAGES_FOLDER = './resources/images'

PHOTOS_FILES = ('pickaxe.png', 'message.png', 'bomb.png')
MAP_TEST = 'map_test.txt'

GRID_SIZE = 500
APP_FONT = 'Helvetica'
TEXT_SIZE = 16
PySimpleGUI.theme('DarkGrey5')
SCORE_STEP = 10

MINEFOL_ASSUMPTIONS = [
    'formulas(assumptions).', 'all x all y (safe(x,y) <-> -(mine(x,y))).', '
   safe(1,1).', 'end_of_list.', '\n']
MINEFOL_GOALS = ['formulas(goals).', '', 'end_of_list.', '\n']

PROVER_INPUT = 'prover9.in'
PROVER_OUTPUT = 'prover9.out'

READ_MODE = 'r'
WRITE_MODE = 'w'

PROVER_COMMAND = f'prover9 -f {PROVER_INPUT} > {PROVER_OUTPUT}'
THEOREM_PROVED = 'THEOREM PROVED'


class Minefield:
    def __init__(self):
        pass

    def initialize(self):
        self.read_config(os.path.join(MAPS_FOLDER, MAP_TEST))
```

```
45
46          self.initialize_window()
47
48          self.initialize_game()
49
50          self.load_pictures()
51
52      def read_config(self, map_path):
53          self.initialize_array()
54
55          row = 0
56          reading_map = True
57          with open(map_path, 'r') as map_file:
58              for line in map_file.readlines():
59                  if line.startswith('='):
60                      reading_map = False
61                      continue
62
63                  if reading_map:
64                      for column in range(len(line)):
65                          if line[column] == 'M':
66                              self.messages[(column, row)] = ''
67                          elif line[column] == 'B':
68                              self.bombs.append((column, row))
69                          elif line[column] == '%':
70                              self.walls.append((column, row))
71
72                      row += 1
73
74                  else:
75                      if line.endswith('\n'):
76                          line = line[:-1]
77
78                      words = line.split(' ')
79                      self.messages[(int(words[0]), int(
80                          words[1]))] = ' '.join(words[2:])
81
82          self.cell_count = row
83          self.cell_size = GRID_SIZE // self.cell_count
84
85      def initialize_array(self):
86          self.messages = {}
87          self.bombs = []
88          self.walls = []
89
90      def load_pictures(self):
91          self.pictures = []
92
93          for resource in PHOTOS_FILES:
94              image = Image.open(os.path.join(IMAGES_FOLDER, resource))
95              image = image.resize(
96                  (self.cell_size, self.cell_size), Image.ANTIALIAS)
97              photoImage = ImageTk.PhotoImage(image)
98              self.pictures.append(photoImage)
99
100     def initialize_matrix(self):
101         self.cell_map = numpy.zeros(
102             (self.cell_count, self.cell_count), dtype=int)
103         self.visited_map = numpy.zeros(
104             (self.cell_count, self.cell_count), dtype=int)
```

```
105         self.safe_map = numpy.zeros(
106             (self.cell_count, self.cell_count), dtype=int)
107 
108     def draw_map(self, xPos, yPos):
109         number_coordinate = self.cell_size // 2 + 2
110 
111         for idx in range(self.cell_count):
112             self.draw_cell(idx, 0, 'GRAY')
113             self.canvas.TKCanvas.create_text(idx * self.cell_size +
    number_coordinate, number_coordinate,
114                                              fill='WHITE', font=' '.join([
    APP_FONT, str(self.cell_size)]),
115                                              text=f'{idx}')
116 
117             self.draw_cell(0, idx, 'GRAY')
118             self.canvas.TKCanvas.create_text(number_coordinate, idx * self.
    cell_size + number_coordinate,
119                                              fill='WHITE', font=' '.join([
    APP_FONT, str(self.cell_size)]),
120                                              text=f'{idx}')
121 
122         for row in range(1, self.cell_count):
123             for col in range(1, self.cell_count):
124                 if (row, col) in self.walls:
125                     self.draw_cell(row, col, 'GRAY')
126 
127                 if self.visited_map[row][col] == 1:
128                     self.draw_cell(row, col)
129 
130                     if (row, col) in self.messages.keys():
131                         self.draw_cell(row, col, 'GREEN')
132                         self.draw_image(row, col, self.pictures[1])
133                     if (row, col) in self.bombs:
134                         self.draw_cell(row, col, 'RED')
135                         self.draw_image(row, col, self.pictures[2])
136 
137         self.draw_image(xPos,
138                         yPos, self.pictures[0])
139 
140     def draw_image(self, x, y, resource):
141         x *= self.cell_size
142         y *= self.cell_size
143 
144         self.canvas.TKCanvas.create_image(
145             x, y, image=resource, anchor='nw')
146 
147     def draw_cell(self, x, y, color='YELLOW'):
148         x *= self.cell_size
149         y *= self.cell_size
150 
151         self.canvas.TKCanvas.create_rectangle(
152             x, y, x + self.cell_size, y + self.cell_size,
153             outline='BLACK', fill=color, width=1)
154 
155     def run(self):
156         while True:
157             self.canvas.TKCanvas.delete('all')
158 
159             self.window['-SCORE-'].update(f'Score: {self.score}')
160 
```

```
            self.draw_grid()

            xPos = self.player_pos[0] // self.cell_size
            yPos = self.player_pos[1] // self.cell_size

            self.visited_map[xPos][yPos] = 1

            self.draw_map(xPos, yPos)

            if self.process_events(xPos, yPos) == -1:
                break

        self.window.close()

    def get_event(self, event):
        move = event
        if event.startswith('Up'):
            move = 'Up'
        elif event.startswith('Down'):
            move = 'Down'
        elif event.startswith('Left'):
            move = 'Left'
        elif event.startswith('Right'):
            move = 'Right'

        return move

    def check_move(self, oldX, oldY, newX, newY):
        if oldX == newX and oldY == newY:
            return

        if (newX, newY) in self.bombs:
            self.window['-MESSAGE-'].update(f'YOU LOST!')
            self.is_running = False

        if (newX, newY) in self.messages.keys():
            current_message = self.messages[(newX, newY)]

            self.window['-MESSAGE-'].update(
                f'{current_message}')

            if current_message not in MINEFOL_ASSUMPTIONS:
                MINEFOL_ASSUMPTIONS.insert(3, current_message)
        else:
            self.window['-MESSAGE-'].update('')

        if not self.visited_map[newX][newY]:
            self.safe_map[newX][newY] = self.check_safe(newX, newY)
            self.score += ((-1) **
                           (not self.safe_map[newX][newY])) * SCORE_STEP
            self.visited_cells += 1

        if self.visited_cells == self.cell_count ** 2 - len(self.bombs) -
    len(self.walls) - 1:
            self.window['-MESSAGE-'].update(f'YOU WON!')
            self.is_running = False

    def check_safe(self, x, y):
        MINEFOL_GOALS[1] = f'safe({x}, {y}).'

```

```
        with open(PROVER_INPUT, WRITE_MODE) as prover_file:
            prover_file.write('\n'.join(MINEFOL_ASSUMPTIONS))
            prover_file.write('\n'.join(MINEFOL_GOALS))

        os.system(PROVER_COMMAND)

        with open(PROVER_OUTPUT, READ_MODE) as prover_file:
            demonstration = prover_file.read()

        return THEOREM_PROVED in demonstration


def main():
    minefield = Minefield()
    minefield.initialize()
    minefield.run()


if __name__ == '__main__':
    main()


#map1

%%%%%%%%%
%M..MM.M.
%........
%..M.BM..
%.B.....M
%......B.
%M..B.M..
%........
%..M....M
=========
1 1 -(exists y mine(y, 1)).
4 1 mine(4, 6).
5 1 all x (mine(1, x) <-> mine(x, 1)).
7 1 exists x (mine(3, x) ->  mine(8, x)).
1 6 all x (mine(x, 8) -> mine(x, 1)).
3 8 -(exists x (mine(x, x))).
8 8 all x all y all z ((mine(y, x) & y!=z)<->-mine(z, x)).
6 6 -mine(2, 3) | -mine(3, 2).
3 3 exists y exists x(mine(y, 6) -> -mine(8, x)).
8 4 mine(3, 5) | mine(5, 3).
6 3 mine(2, 4) & mine(7, 5).


#map2

%%%%%%%%%
%M.B.....
%.M.B....
%.......M
%...MBB..
%.M.MM..M
%M.....M.
%M.......
%.B...M.M
=========
1 1 all x (-mine(1,x)).
```

```
1 6 all y (-mine(y,3)).
1 7 mine(6,4).
8 3 exists x (mine(1,x) -> mine(8,x)).
8 5 (-mine(4,5) | -mine(5, 4)).
8 8 all x(-mine(x, x)).
2 2 all x(-mine(x,7)).
4 4 all x all y all z ((mine(y,x) & x!=z)<->-mine(y,z)).
6 5 mine(5,4).
6 8 mine(2,8).
2 5 all x(-mine(7,x)).
7 6 (mine(4,2) | mine(5,1)).
4 5 mine(3,1).

#map3

%%%%%%%%%
%MBMBMM..
%.B.MM..B
%.......M
%..M..MMB
%......M.
%M.....M.
%.BM.....
%M....M..
=========
1 1 all x (-mine(1,x)).
1 6 all y (-mine(y,3)).
1 8 -mine(3,1) & -mine(3,2).
8 3 all y (safe(y,8)).
3 1 safe(4,2).
4 2 all y(-mine(y,5)).
6 8 all x(-mine(6,x)).
6 4 mine(8,4).
6 1 -mine(5,1).
7 5 safe(7,1)&safe(8,1)&safe(7,2).
5 1 all y(-mine(y,6)).
7 6 all x (-mine(5,x)).
5 2 exists x all y(mine(8, x) <-> -mine(3, y)).
3 4 mine(2,7).
3 7 -mine(2,4)&-mine(4,4)&-mine(7,4).
7 4 exists x all y((mine(x,7)&x!=y) <-> (-mine(y,7))).
```