Міністерство освіти і науки України

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Програмування інтелектуальних інформаційних систем

**ЗВІТ**

до лабораторних робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | ІТ-92, Бондаренко Дмитро Сергійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Баришич Л.М. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

# **Завдання лабораторної роботи**

Темою цієї роботи є створення основи для проекту, над яким подалі буде вестися робота. У цій роботі буде створена гра Pac-Man. У цій грі буде повністю покритий функціонал для одного гравця.

# **Опис використаних технологій**

У цій роботі буде використана мова програмування Python, оскільки вона має зручну бібліотеку для відтворення графічного інтерфейсу, а й власне гри, tkinter.

# **Опис програмного коду**

У нашому проекті є 9 файлів:

game.py – загальний рушій гри. У ньому є описи правил, абстрактний клас агентів гри (Пак-ман, Привиди), та метод який запускає саму гру.

ghostAgents.py – агенти привидів, тобто та поведінка, яку будуть мати привиди.

graphicsDisplay.py – рендеринг самої гри (ігрове поле).

graphicsUtils.py – допоміжні методи для рендерингу гри та опрацювання натискань клавіш.

keyboardAgent.py – агенти, які мають опрацьовуватися з клавіатури.

layout.py – парсинг поля, розмітку якого ми маємо написати самі.

pacman.py – вхідна точка для роботи програми. Тут також маються правила гри, та парсинг аргументів консолі.

util.py – допоміжні методи для роботи гри.

originalClassic.lay – класична розмітка поля для гри.

Розглянемо кожний файл більш детально.

**game.py**

from util import raiseNotDefined  
  
  
# Абстрактний метод для агентів  
class Agent:  
  
 # Агент буде отримувати стан гри та повертати дію із шляху  
 def getAction(self, state):  
 raiseNotDefined()  
  
 def \_\_init\_\_(self, index=0):  
 self.index = index  
  
  
# Шляхи для пересування агентів  
class Directions:  
 NORTH = 'North'  
 SOUTH = 'South'  
 EAST = 'East'  
 WEST = 'West'  
 STOP = 'Stop'  
  
 LEFT = {NORTH: WEST,  
 SOUTH: EAST,  
 EAST: NORTH,  
 WEST: SOUTH,  
 STOP: STOP}  
  
 RIGHT = dict([(y, x) for x, y in LEFT.items()])  
  
 REVERSE = {NORTH: SOUTH,  
 SOUTH: NORTH,  
 EAST: WEST,  
 WEST: EAST,  
 STOP: STOP}  
  
  
# Конфігурація містить координати (x, y) персонажа разом з його напрямоком пересування.  
class Configuration:  
  
 def \_\_init\_\_(self, pos, direction):  
 self.pos = pos  
 self.direction = direction  
  
 # Отримати позицію персонажа  
 def getPosition(self):  
 return self.pos  
  
 # Отримати напрямок персонажа  
 def getDirection(self):  
 return self.direction  
  
 # Створює нову конфігурацію, досягнуту шляхом перекладу поточної конфігурація за вектором дії.  
 def generateSuccessor(self, vector):  
 x, y = self.pos  
 dx, dy = vector  
 direction = Actions.vectorToDirection(vector)  
 if direction == Directions.STOP:  
 direction = self.direction # There is no stop direction  
 return Configuration((x + dx, y + dy), direction)  
  
  
# Утримує стан агента (конфігурація, швидкість, наляканість тощо).  
class AgentState:  
 def \_\_init\_\_(self, startConfiguration, isPacman):  
 self.start = startConfiguration  
 self.configuration = startConfiguration  
 self.isPacman = isPacman  
 self.scaredTimer = 0  
 self.numCarrying = 0  
 self.numReturned = 0  
  
 def copy(self):  
 state = AgentState(self.start, self.isPacman)  
 state.configuration = self.configuration  
 state.scaredTimer = self.scaredTimer  
 state.numCarrying = self.numCarrying  
 state.numReturned = self.numReturned  
 return state  
  
 # Отримати позицію агента  
 def getPosition(self):  
 if self.configuration is None:  
 return None  
 return self.configuration.getPosition()  
  
 # Отримати напрямок агента  
 def getDirection(self):  
 return self.configuration.getDirection()  
  
  
# Сітка деяких об'єктів. Двовимірний масив об’єктів, підкріплений списком списків.  
class Grid:  
 def \_\_init\_\_(self, width, height, initialValue=False):  
 if initialValue not in [False, True]: raise Exception('Grids can only contain booleans')  
 self.CELLS\_PER\_INT = 30  
  
 self.width = width  
 self.height = height  
 self.data = [[initialValue for y in range(height)] for x in range(width)]  
  
 def \_\_getitem\_\_(self, i):  
 return self.data[i]  
  
 def copy(self):  
 g = Grid(self.width, self.height)  
 g.data = [x[:] for x in self.data]  
 return g  
  
 def deepCopy(self):  
 return self.copy()  
  
 def shallowCopy(self):  
 g = Grid(self.width, self.height)  
 g.data = self.data  
 return g  
  
 # Рахунок об'єктів сітки  
 def count(self, item=True):  
 return sum([x.count(item) for x in self.data])  
  
 # Конвертація сітки у список  
 def asList(self, key=True):  
 list = []  
 for x in range(self.width):  
 for y in range(self.height):  
 if self[x][y] == key: list.append((x, y))  
 return list  
  
  
# Колекція статичних методів для маніпулювання діями переміщення.  
class Actions:  
 # Шляхи  
 \_directions = {Directions.NORTH: (0, 1),  
 Directions.SOUTH: (0, -1),  
 Directions.EAST: (1, 0),  
 Directions.WEST: (-1, 0),  
 Directions.STOP: (0, 0)}  
  
 \_directionsAsList = \_directions.items()  
  
 TOLERANCE = .001  
  
 # Метод, який перевертає шляхи. Потрібен для того, щоб привиди пересувалися більш природньо  
 def reverseDirection(action):  
 if action == Directions.NORTH:  
 return Directions.SOUTH  
 if action == Directions.SOUTH:  
 return Directions.NORTH  
 if action == Directions.EAST:  
 return Directions.WEST  
 if action == Directions.WEST:  
 return Directions.EAST  
 return action  
  
 reverseDirection = staticmethod(reverseDirection)  
  
 # Конвертація вектору у шлях  
 def vectorToDirection(vector):  
 dx, dy = vector  
 if dy > 0:  
 return Directions.NORTH  
 if dy < 0:  
 return Directions.SOUTH  
 if dx < 0:  
 return Directions.WEST  
 if dx > 0:  
 return Directions.EAST  
 return Directions.STOP  
  
 vectorToDirection = staticmethod(vectorToDirection)  
  
 # Конвертація шляху у вектор  
 def directionToVector(direction, speed=1.0):  
 dx, dy = Actions.\_directions[direction]  
 return dx \* speed, dy \* speed  
  
 directionToVector = staticmethod(directionToVector)  
  
 # Метод для отримання можливих дій для агента  
 def getPossibleActions(config, walls):  
 possible = []  
 x, y = config.pos  
 x\_int, y\_int = int(x + 0.5), int(y + 0.5)  
  
 if abs(x - x\_int) + abs(y - y\_int) > Actions.TOLERANCE:  
 return [config.getDirection()]  
  
 for dir, vec in Actions.\_directionsAsList:  
 dx, dy = vec  
 next\_y = y\_int + dy  
 next\_x = x\_int + dx  
 if not walls[next\_x][next\_y]: possible.append(dir)  
  
 return possible  
  
 getPossibleActions = staticmethod(getPossibleActions)  
  
 # Метод для отримання легальних сусідів. Потрібен, щоб привиди не сливалися один із одним  
 def getLegalNeighbors(position, walls):  
 x, y = position  
 x\_int, y\_int = int(x + 0.5), int(y + 0.5)  
 neighbors = []  
 for dir, vec in Actions.\_directionsAsList:  
 dx, dy = vec  
 next\_x = x\_int + dx  
 if next\_x < 0 or next\_x == walls.width:  
 continue  
 next\_y = y\_int + dy  
 if next\_y < 0 or next\_y == walls.height:  
 continue  
 if not walls[next\_x][next\_y]: neighbors.append((next\_x, next\_y))  
 return neighbors  
  
 getLegalNeighbors = staticmethod(getLegalNeighbors)  
  
  
# Створює новий пакет даних шляхом копіювання інформації з попередника.  
class GameStateData:  
 def \_\_init\_\_(self, prevState=None):  
 if prevState is not None:  
 self.food = prevState.food.shallowCopy()  
 self.capsules = prevState.capsules[:]  
 self.agentStates = self.copyAgentStates(prevState.agentStates)  
 self.layout = prevState.layout  
 self.\_eaten = prevState.\_eaten  
 self.score = prevState.score  
  
 self.\_foodEaten = None  
 self.\_foodAdded = None  
 self.\_capsuleEaten = None  
 self.\_agentMoved = None  
 self.\_lose = False  
 self.\_win = False  
 self.scoreChange = 0  
  
 def deepCopy(self):  
 state = GameStateData(self)  
 state.food = self.food.deepCopy()  
 state.layout = self.layout.deepCopy()  
 state.\_agentMoved = self.\_agentMoved  
 state.\_foodEaten = self.\_foodEaten  
 state.\_foodAdded = self.\_foodAdded  
 state.\_capsuleEaten = self.\_capsuleEaten  
 return state  
  
 # Метод для копіювання статусів агента  
 def copyAgentStates(self, agentStates):  
 copiedStates = []  
 for agentState in agentStates:  
 copiedStates.append(agentState.copy())  
 return copiedStates  
  
 # Створює початковий стан гри з масиву макета  
 def initialize(self, layout, numGhostAgents):  
 self.food = layout.food.copy()  
 self.capsules = layout.capsules[:]  
 self.layout = layout  
 self.score = 0  
 self.scoreChange = 0  
  
 self.agentStates = []  
 numGhosts = 0  
 for isPacman, pos in layout.agentPositions:  
 if not isPacman:  
 if numGhosts == numGhostAgents:  
 continue  
 else:  
 numGhosts += 1  
 self.agentStates.append(AgentState(Configuration(pos, Directions.STOP), isPacman))  
 self.\_eaten = [False for a in self.agentStates]  
  
  
# Гра керує потоком управління, вимагаючи дії від агентів.  
class Game:  
 def \_\_init\_\_(self, agents, display, rules, startingIndex=0, muteAgents=False, catchExceptions=False):  
 self.state = None  
 self.agentCrashed = False  
 self.agents = agents  
 self.display = display  
 self.rules = rules  
 self.startingIndex = startingIndex  
 self.gameOver = False  
 self.muteAgents = muteAgents  
 self.catchExceptions = catchExceptions  
 self.moveHistory = []  
 self.totalAgentTimes = [0 for agent in agents]  
 self.totalAgentTimeWarnings = [0 for agent in agents]  
 self.agentTimeout = False  
 import io  
 self.agentOutput = [io.StringIO() for agent in agents]  
  
 # Основний цикл для того, щоб грати  
 def run(self):  
 self.display.initialize(self.state.data)  
  
 agentIndex = self.startingIndex  
 numAgents = len(self.agents)  
  
 while not self.gameOver:  
 agent = self.agents[agentIndex]  
 if 'observationFunction' in dir(agent):  
 observation = agent.observationFunction(self.state.deepCopy())  
 else:  
 observation = self.state.deepCopy()  
 action = agent.getAction(observation)  
 self.moveHistory.append((agentIndex, action))  
 self.state = self.state.generateSuccessor(agentIndex, action)  
 self.display.update(self.state.data)  
 self.rules.process(self.state, self)  
 agentIndex = (agentIndex + 1) % numAgents

**ghostAgents.py**

from game import Agent  
import util  
  
  
# Абстрактний клас для агентів привидів  
class GhostAgent(Agent):  
 def \_\_init\_\_(self, index):  
 super().\_\_init\_\_(index)  
 self.index = index  
  
 # Метод для отримання дії від агента  
 def getAction(self, state):  
 dist = self.getDistribution(state)  
 return util.chooseFromDistribution(dist)  
  
 # Повертає лічильник, що кодує розподіл над діями з наданого стану.  
 def getDistribution(self, state):  
 *"Returns a Counter encoding a distribution over actions from the provided state."* util.raiseNotDefined()  
  
  
# Привид, який обирає наступну дію рівномірно випадковим чином.  
class RandomGhost(GhostAgent):  
  
 def getDistribution(self, state):  
 dist = util.Counter()  
 for a in state.getLegalActions(self.index): dist[a] = 1.0  
 dist.normalize()  
 return dist

**graphichDisplay.py**

from game import Directions  
from graphicsUtils import \*  
  
# Налаштування розмірів та кольорів  
DEFAULT\_GRID\_SIZE = 23.0  
INFO\_PANE\_HEIGHT = 35  
BACKGROUND\_COLOR = formatColor(0, 0, 0)  
WALL\_COLOR = formatColor(.9, 0, 0)  
INFO\_PANE\_COLOR = formatColor(.4, .4, 0)  
SCORE\_COLOR = formatColor(.9, .9, .9)  
PACMAN\_OUTLINE\_WIDTH = 2  
PACMAN\_CAPTURE\_OUTLINE\_WIDTH = 4  
GHOST\_COLORS = []  
GHOST\_COLORS.append(formatColor(.9, 0, 0)) # Red  
GHOST\_COLORS.append(formatColor(0, .3, .9)) # Blue  
GHOST\_COLORS.append(formatColor(.98, .41, .07)) # Orange  
GHOST\_COLORS.append(formatColor(.1, .75, .7)) # Green  
GHOST\_COLORS.append(formatColor(1.0, 0.6, 0.0)) # Yellow  
GHOST\_COLORS.append(formatColor(.4, 0.13, 0.91)) # Purple  
TEAM\_COLORS = GHOST\_COLORS[:2]  
GHOST\_SHAPE = [  
 (0, 0.3),  
 (0.25, 0.75),  
 (0.5, 0.3),  
 (0.75, 0.75),  
 (0.75, -0.5),  
 (0.5, -0.75),  
 (-0.5, -0.75),  
 (-0.75, -0.5),  
 (-0.75, 0.75),  
 (-0.5, 0.3),  
 (-0.25, 0.75)  
]  
GHOST\_SIZE = 0.65  
SCARED\_COLOR = formatColor(1, 1, 1)  
GHOST\_VEC\_COLORS = [colorToVector(c) for c in GHOST\_COLORS]  
PACMAN\_COLOR = formatColor(255.0 / 255.0, 255.0 / 255.0, 61.0 / 255)  
PACMAN\_SCALE = 0.5  
FOOD\_COLOR = formatColor(1, 1, 1)  
FOOD\_SIZE = 0.1  
CAPSULE\_COLOR = formatColor(1, 1, 1)  
CAPSULE\_SIZE = 0.25  
WALL\_RADIUS = 0.25  
  
# Відмальовка панелі з інформацією  
class InfoPane:  
 def \_\_init\_\_(self, layout, gridSize):  
 self.gridSize = gridSize  
 self.width = layout.width \* gridSize  
 self.base = (layout.height + 1) \* gridSize  
 self.height = INFO\_PANE\_HEIGHT  
 self.fontSize = 24  
 self.textColor = PACMAN\_COLOR  
 self.drawPane()  
  
 # Переносимо відносну точку з нижнього лівого кута інформаційної області.  
 def toScreen(self, pos, y=None):  
 if y is None:  
 x, y = pos  
 else:  
 x = pos  
  
 x = self.gridSize + x  
 y = self.base + y  
 return x, y  
  
 # Рендеринг панелі із інформацією про рахунок гри  
 def drawPane(self):  
 self.scoreText = text(self.toScreen(0, 0), self.textColor, "SCORE: 0", "Times", self.fontSize, "bold")  
  
 # Метод для оновлення рахунку гри на панелі  
 def updateScore(self, score):  
 changeText(self.scoreText, "SCORE: % 4d" % score)  
  
  
# Графіка  
class PacmanGraphics:  
 def \_\_init\_\_(self, zoom=1.0, frameTime=0.0, capture=False):  
 self.have\_window = 0  
 self.currentGhostImages = {}  
 self.pacmanImage = None  
 self.zoom = zoom  
 self.gridSize = DEFAULT\_GRID\_SIZE \* zoom  
 self.capture = capture  
 self.frameTime = frameTime  
  
 def initialize(self, state):  
 self.startGraphics(state)  
 self.distributionImages = None  
 self.drawStaticObjects(state)  
 self.drawAgentObjects(state)  
  
 # Початок рендерингу графіки  
 def startGraphics(self, state):  
 self.layout = state.layout  
 layout = self.layout  
 self.width = layout.width  
 self.height = layout.height  
 self.make\_window(self.width, self.height)  
 self.infoPane = InfoPane(layout, self.gridSize)  
 self.currentState = layout  
  
 # Рендеринг статичних об'єктів  
 def drawStaticObjects(self, state):  
 layout = self.layout  
 self.drawWalls(layout.walls)  
 self.food = self.drawFood(layout.food)  
 self.capsules = self.drawCapsules(layout.capsules)  
  
 # Рендеринг об'єктів агентів  
 def drawAgentObjects(self, state):  
 self.agentImages = [] # (agentState, image)  
 for index, agent in enumerate(state.agentStates):  
 if agent.isPacman:  
 image = self.drawPacman(agent, index)  
 self.agentImages.append((agent, image))  
 else:  
 image = self.drawGhost(agent, index)  
 self.agentImages.append((agent, image))  
  
 # Змінює зображення з привида на pac-man або навпаки (для захоплення)  
 def swapImages(self, agentIndex, newState):  
 prevState, prevImage = self.agentImages[agentIndex]  
 for item in prevImage: remove\_from\_screen(item)  
 if newState.isPacman:  
 image = self.drawPacman(newState, agentIndex)  
 self.agentImages[agentIndex] = (newState, image)  
 else:  
 image = self.drawGhost(newState, agentIndex)  
 self.agentImages[agentIndex] = (newState, image)  
  
 # Оновлення дисплею  
 def update(self, newState):  
 agentIndex = newState.\_agentMoved  
 agentState = newState.agentStates[agentIndex]  
  
 if self.agentImages[agentIndex][0].isPacman != agentState.isPacman: self.swapImages(agentIndex, agentState)  
 prevState, prevImage = self.agentImages[agentIndex]  
 if agentState.isPacman:  
 self.animatePacman(agentState, prevState, prevImage)  
 else:  
 self.moveGhost(agentState, agentIndex, prevState, prevImage)  
 self.agentImages[agentIndex] = (agentState, prevImage)  
  
 if newState.\_foodEaten != None:  
 self.removeFood(newState.\_foodEaten, self.food)  
 if newState.\_capsuleEaten != None:  
 self.removeCapsule(newState.\_capsuleEaten, self.capsules)  
 self.infoPane.updateScore(newState.score)  
  
 # Створення вікна для гри  
 def make\_window(self, width, height):  
 grid\_width = (width - 1) \* self.gridSize  
 grid\_height = (height - 1) \* self.gridSize  
 screen\_width = 2 \* self.gridSize + grid\_width  
 screen\_height = 2 \* self.gridSize + grid\_height + INFO\_PANE\_HEIGHT  
  
 begin\_graphics(screen\_width,  
 screen\_height,  
 BACKGROUND\_COLOR,  
 "Pacman")  
  
 # Рендеринг Pac-man  
 def drawPacman(self, pacman, index):  
 position = self.getPosition(pacman)  
 screen\_point = self.to\_screen(position)  
 endpoints = self.getEndpoints(self.getDirection(pacman))  
  
 width = PACMAN\_OUTLINE\_WIDTH  
 outlineColor = PACMAN\_COLOR  
 fillColor = PACMAN\_COLOR  
  
 if self.capture:  
 outlineColor = TEAM\_COLORS[index % 2]  
 fillColor = GHOST\_COLORS[index]  
 width = PACMAN\_CAPTURE\_OUTLINE\_WIDTH  
  
 return [circle(screen\_point, PACMAN\_SCALE \* self.gridSize,  
 fillColor=fillColor, outlineColor=outlineColor,  
 endpoints=endpoints,  
 width=width)]  
  
 # Рендеринг Pac-man в залежності від сторони руху  
 def getEndpoints(self, direction, position=(0, 0)):  
 x, y = position  
 pos = x - int(x) + y - int(y)  
 width = 30 + 80 \* math.sin(math.pi \* pos)  
  
 delta = width / 2  
 if direction == 'West':  
 endpoints = (180 + delta, 180 - delta)  
 elif direction == 'North':  
 endpoints = (90 + delta, 90 - delta)  
 elif direction == 'South':  
 endpoints = (270 + delta, 270 - delta)  
 else:  
 endpoints = (0 + delta, 0 - delta)  
 return endpoints  
  
 # Пересування Pac-man  
 def movePacman(self, position, direction, image):  
 screenPosition = self.to\_screen(position)  
 endpoints = self.getEndpoints(direction, position)  
 r = PACMAN\_SCALE \* self.gridSize  
 moveCircle(image[0], screenPosition, r, endpoints)  
  
 # Анімація для Pac-man  
 def animatePacman(self, pacman, prevPacman, image):  
 if self.frameTime < 0:  
 print('Press any key to step forward, "q" to play')  
 keys = wait\_for\_keys()  
 if 'q' in keys:  
 self.frameTime = 0.1  
 if self.frameTime > 0.01 or self.frameTime < 0:  
 fx, fy = self.getPosition(prevPacman)  
 px, py = self.getPosition(pacman)  
 frames = 4.0  
 for i in range(1, int(frames) + 1):  
 pos = px \* i / frames + fx \* (frames - i) / frames, py \* i / frames + fy \* (frames - i) / frames  
 self.movePacman(pos, self.getDirection(pacman), image)  
  
 sleep(abs(self.frameTime) / frames)  
 else:  
 self.movePacman(self.getPosition(pacman), self.getDirection(pacman), image)  
  
 # Метот для отримання кольору привида  
 def getGhostColor(self, ghost, ghostIndex):  
 if ghost.scaredTimer > 0:  
 return SCARED\_COLOR  
 else:  
 return GHOST\_COLORS[ghostIndex]  
  
 # Метод для рендерингу привида  
 def drawGhost(self, ghost, agentIndex):  
 pos = self.getPosition(ghost)  
 dir = self.getDirection(ghost)  
 (screen\_x, screen\_y) = (self.to\_screen(pos) )  
 coords = []  
 for (x, y) in GHOST\_SHAPE:  
 coords.append((x\*self.gridSize\*GHOST\_SIZE + screen\_x, y\*self.gridSize\*GHOST\_SIZE + screen\_y))  
  
 colour = self.getGhostColor(ghost, agentIndex)  
 body = polygon(coords, colour, filled = 1)  
 WHITE = formatColor(1.0, 1.0, 1.0)  
 BLACK = formatColor(0.0, 0.0, 0.0)  
  
 dx = 0  
 dy = 0  
 if dir == 'North':  
 dy = -0.2  
 if dir == 'South':  
 dy = 0.2  
 if dir == 'East':  
 dx = 0.2  
 if dir == 'West':  
 dx = -0.2  
 leftEye = circle((screen\_x+self.gridSize\*GHOST\_SIZE\*(-0.3+dx/1.5), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy/1.5)), self.gridSize\*GHOST\_SIZE\*0.2, WHITE, WHITE)  
 rightEye = circle((screen\_x+self.gridSize\*GHOST\_SIZE\*(0.3+dx/1.5), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy/1.5)), self.gridSize\*GHOST\_SIZE\*0.2, WHITE, WHITE)  
 leftPupil = circle((screen\_x+self.gridSize\*GHOST\_SIZE\*(-0.3+dx), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy)), self.gridSize\*GHOST\_SIZE\*0.08, BLACK, BLACK)  
 rightPupil = circle((screen\_x+self.gridSize\*GHOST\_SIZE\*(0.3+dx), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy)), self.gridSize\*GHOST\_SIZE\*0.08, BLACK, BLACK)  
 ghostImageParts = []  
 ghostImageParts.append(body)  
 ghostImageParts.append(leftEye)  
 ghostImageParts.append(rightEye)  
 ghostImageParts.append(leftPupil)  
 ghostImageParts.append(rightPupil)  
  
 return ghostImageParts  
  
 # Метод для переміщення очей привида  
 def moveEyes(self, pos, dir, eyes):  
 (screen\_x, screen\_y) = (self.to\_screen(pos) )  
 dx = 0  
 dy = 0  
 if dir == 'North':  
 dy = -0.2  
 if dir == 'South':  
 dy = 0.2  
 if dir == 'East':  
 dx = 0.2  
 if dir == 'West':  
 dx = -0.2  
 moveCircle(eyes[0],(screen\_x+self.gridSize\*GHOST\_SIZE\*(-0.3+dx/1.5), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy/1.5)), self.gridSize\*GHOST\_SIZE\*0.2)  
 moveCircle(eyes[1],(screen\_x+self.gridSize\*GHOST\_SIZE\*(0.3+dx/1.5), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy/1.5)), self.gridSize\*GHOST\_SIZE\*0.2)  
 moveCircle(eyes[2],(screen\_x+self.gridSize\*GHOST\_SIZE\*(-0.3+dx), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy)), self.gridSize\*GHOST\_SIZE\*0.08)  
 moveCircle(eyes[3],(screen\_x+self.gridSize\*GHOST\_SIZE\*(0.3+dx), screen\_y-self.gridSize\*GHOST\_SIZE\*(0.3-dy)), self.gridSize\*GHOST\_SIZE\*0.08)  
  
 # Методя для опрацювання переміщення привида  
 def moveGhost(self, ghost, ghostIndex, prevGhost, ghostImageParts):  
 old\_x, old\_y = self.to\_screen(self.getPosition(prevGhost))  
 new\_x, new\_y = self.to\_screen(self.getPosition(ghost))  
 delta = new\_x - old\_x, new\_y - old\_y  
  
 for ghostImagePart in ghostImageParts:  
 move\_by(ghostImagePart, delta)  
  
 if ghost.scaredTimer > 0:  
 color = SCARED\_COLOR  
 else:  
 color = GHOST\_COLORS[ghostIndex]  
 edit(ghostImageParts[0], ('fill', color), ('outline', color))  
 self.moveEyes(self.getPosition(ghost), self.getDirection(ghost), ghostImageParts[-4:])  
 # Отримати позицію агента  
 def getPosition(self, agentState):  
 if agentState.configuration is None:  
 return -1000, -1000  
 return agentState.getPosition()  
  
 # Отримати напрямок агента  
 def getDirection(self, agentState):  
 if agentState.configuration is None:  
 return Directions.STOP  
 return agentState.configuration.getDirection()  
  
 # Розташування анімованого агента на екрані  
 def to\_screen(self, point):  
 (x, y) = point  
 x = (x + 1) \* self.gridSize  
 y = (self.height - y) \* self.gridSize  
 return x, y  
  
 # Метод для рендерингу стін  
 def drawWalls(self, wallMatrix):  
 wallColor = WALL\_COLOR  
 for xNum, x in enumerate(wallMatrix):  
 if self.capture and (xNum \* 2) < wallMatrix.width: wallColor = TEAM\_COLORS[0]  
 if self.capture and (xNum \* 2) >= wallMatrix.width: wallColor = TEAM\_COLORS[1]  
  
 for yNum, cell in enumerate(x):  
 if cell: # There's a wall here  
 pos = (xNum, yNum)  
 screen = self.to\_screen(pos)  
  
 # draw each quadrant of the square based on adjacent walls  
 wIsWall = self.isWall(xNum - 1, yNum, wallMatrix)  
 eIsWall = self.isWall(xNum + 1, yNum, wallMatrix)  
 nIsWall = self.isWall(xNum, yNum + 1, wallMatrix)  
 sIsWall = self.isWall(xNum, yNum - 1, wallMatrix)  
 nwIsWall = self.isWall(xNum - 1, yNum + 1, wallMatrix)  
 swIsWall = self.isWall(xNum - 1, yNum - 1, wallMatrix)  
 neIsWall = self.isWall(xNum + 1, yNum + 1, wallMatrix)  
 seIsWall = self.isWall(xNum + 1, yNum - 1, wallMatrix)  
  
 # NE quadrant  
 if (not nIsWall) and (not eIsWall):  
 # inner circle  
 circle(screen, WALL\_RADIUS \* self.gridSize, wallColor, wallColor, (0, 91), 'arc')  
 if nIsWall and (not eIsWall):  
 # vertical line  
 line(add(screen, (self.gridSize \* WALL\_RADIUS, 0)),  
 add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* (-0.5) - 1)), wallColor)  
 if (not nIsWall) and eIsWall:  
 # horizontal line  
 line(add(screen, (0, self.gridSize \* (-1) \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* 0.5 + 1, self.gridSize \* (-1) \* WALL\_RADIUS)), wallColor)  
 if nIsWall and eIsWall and (not neIsWall):  
 # outer circle  
 circle(add(screen, (self.gridSize \* 2 \* WALL\_RADIUS, self.gridSize \* (-2) \* WALL\_RADIUS)),  
 WALL\_RADIUS \* self.gridSize - 1, wallColor, wallColor, (180, 271), 'arc')  
 line(add(screen, (self.gridSize \* 2 \* WALL\_RADIUS - 1, self.gridSize \* (-1) \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* 0.5 + 1, self.gridSize \* (-1) \* WALL\_RADIUS)), wallColor)  
 line(add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* (-2) \* WALL\_RADIUS + 1)),  
 add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* (-0.5))), wallColor)  
  
 # NW quadrant  
 if (not nIsWall) and (not wIsWall):  
 # inner circle  
 circle(screen, WALL\_RADIUS \* self.gridSize, wallColor, wallColor, (90, 181), 'arc')  
 if nIsWall and (not wIsWall):  
 # vertical line  
 line(add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, 0)),  
 add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* (-0.5) - 1)), wallColor)  
 if (not nIsWall) and wIsWall:  
 # horizontal line  
 line(add(screen, (0, self.gridSize \* (-1) \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* (-0.5) - 1, self.gridSize \* (-1) \* WALL\_RADIUS)), wallColor)  
 if nIsWall and wIsWall and (not nwIsWall):  
 # outer circle  
 circle(add(screen, (self.gridSize \* (-2) \* WALL\_RADIUS, self.gridSize \* (-2) \* WALL\_RADIUS)),  
 WALL\_RADIUS \* self.gridSize - 1, wallColor, wallColor, (270, 361), 'arc')  
 line(add(screen, (self.gridSize \* (-2) \* WALL\_RADIUS + 1, self.gridSize \* (-1) \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* (-0.5), self.gridSize \* (-1) \* WALL\_RADIUS)), wallColor)  
 line(add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* (-2) \* WALL\_RADIUS + 1)),  
 add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* (-0.5))), wallColor)  
  
 # SE quadrant  
 if (not sIsWall) and (not eIsWall):  
 # inner circle  
 circle(screen, WALL\_RADIUS \* self.gridSize, wallColor, wallColor, (270, 361), 'arc')  
 if sIsWall and (not eIsWall):  
 # vertical line  
 line(add(screen, (self.gridSize \* WALL\_RADIUS, 0)),  
 add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* 0.5 + 1)), wallColor)  
 if (not sIsWall) and eIsWall:  
 # horizontal line  
 line(add(screen, (0, self.gridSize \* 1 \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* 0.5 + 1, self.gridSize \* 1 \* WALL\_RADIUS)), wallColor)  
 if sIsWall and eIsWall and (not seIsWall):  
 # outer circle  
 circle(add(screen, (self.gridSize \* 2 \* WALL\_RADIUS, self.gridSize \* 2 \* WALL\_RADIUS)),  
 WALL\_RADIUS \* self.gridSize - 1, wallColor, wallColor, (90, 181), 'arc')  
 line(add(screen, (self.gridSize \* 2 \* WALL\_RADIUS - 1, self.gridSize \* 1 \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* 0.5, self.gridSize \* 1 \* WALL\_RADIUS)), wallColor)  
 line(add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* 2 \* WALL\_RADIUS - 1)),  
 add(screen, (self.gridSize \* WALL\_RADIUS, self.gridSize \* 0.5)), wallColor)  
  
 # SW quadrant  
 if (not sIsWall) and (not wIsWall):  
 # inner circle  
 circle(screen, WALL\_RADIUS \* self.gridSize, wallColor, wallColor, (180, 271), 'arc')  
 if sIsWall and (not wIsWall):  
 # vertical line  
 line(add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, 0)),  
 add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* 0.5 + 1)), wallColor)  
 if (not sIsWall) and wIsWall:  
 # horizontal line  
 line(add(screen, (0, self.gridSize \* 1 \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* (-0.5) - 1, self.gridSize \* 1 \* WALL\_RADIUS)), wallColor)  
 if sIsWall and wIsWall and (not swIsWall):  
 # outer circle  
 circle(add(screen, (self.gridSize \* (-2) \* WALL\_RADIUS, self.gridSize \* 2 \* WALL\_RADIUS)),  
 WALL\_RADIUS \* self.gridSize - 1, wallColor, wallColor, (0, 91), 'arc')  
 line(add(screen, (self.gridSize \* (-2) \* WALL\_RADIUS + 1, self.gridSize \* 1 \* WALL\_RADIUS)),  
 add(screen, (self.gridSize \* (-0.5), self.gridSize \* 1 \* WALL\_RADIUS)), wallColor)  
 line(add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* 2 \* WALL\_RADIUS - 1)),  
 add(screen, (self.gridSize \* (-1) \* WALL\_RADIUS, self.gridSize \* 0.5)), wallColor)  
  
 # Метод для перевірки наявності стіни  
 def isWall(self, x, y, walls):  
 if x < 0 or y < 0:  
 return False  
 if x >= walls.width or y >= walls.height:  
 return False  
 return walls[x][y]  
  
 # Рендеринг їжі  
 def drawFood(self, foodMatrix):  
 foodImages = []  
 color = FOOD\_COLOR  
 for xNum, x in enumerate(foodMatrix):  
 if self.capture and (xNum \* 2) <= foodMatrix.width: color = TEAM\_COLORS[0]  
 if self.capture and (xNum \* 2) > foodMatrix.width: color = TEAM\_COLORS[1]  
 imageRow = []  
 foodImages.append(imageRow)  
 for yNum, cell in enumerate(x):  
 if cell: # There's food here  
 screen = self.to\_screen((xNum, yNum))  
 dot = circle(screen,  
 FOOD\_SIZE \* self.gridSize,  
 outlineColor=color, fillColor=color,  
 width=1)  
 imageRow.append(dot)  
 else:  
 imageRow.append(None)  
 return foodImages  
  
 # Рендеринг капсул  
 def drawCapsules(self, capsules):  
 capsuleImages = {}  
 for capsule in capsules:  
 (screen\_x, screen\_y) = self.to\_screen(capsule)  
 dot = circle((screen\_x, screen\_y),  
 CAPSULE\_SIZE \* self.gridSize,  
 outlineColor=CAPSULE\_COLOR,  
 fillColor=CAPSULE\_COLOR,  
 width=1)  
 capsuleImages[capsule] = dot  
 return capsuleImages  
  
 # Прибрати їжу  
 def removeFood(self, cell, foodImages):  
 x, y = cell  
 remove\_from\_screen(foodImages[x][y])  
  
 # Прибрати капсулу  
 def removeCapsule(self, cell, capsuleImages):  
 x, y = cell  
 remove\_from\_screen(capsuleImages[(x, y)])  
  
  
# Метод для додавання векторів  
def add(x, y):  
 return x[0] + y[0], x[1] + y[1]

**graphichUtil.py**

import os.path  
import sys  
import time  
import tkinter  
import math  
  
# Налаштування полотна для графіки  
\_root\_window = None  
\_canvas = None  
\_canvas\_xs = None  
\_canvas\_ys = None  
\_canvas\_x = None  
\_canvas\_y = None  
\_canvas\_col = None  
\_canvas\_tsize = 12  
\_canvas\_tserifs = 0  
\_canvas\_tfonts = ['times', 'lucidasans-24']  
  
# Конвертація кольору в деякий формат  
def formatColor(r, g, b):  
 return '#%02x%02x%02x' % (int(r \* 255), int(g \* 255), int(b \* 255))  
  
# Конвертація кольору у вектор  
def colorToVector(color):  
 return list(map(lambda x: int(x, 16) / 256.0, [color[1:3], color[3:5], color[5:7]]))  
  
# Функція для очікування. Без неї неможливо закрити гру  
def sleep(secs):  
 global \_root\_window  
 if \_root\_window == None:  
 time.sleep(secs)  
 else:  
 \_root\_window.update\_idletasks()  
 \_root\_window.after(int(1000 \* secs), \_root\_window.quit)  
 \_root\_window.mainloop()  
  
# Початок відмалювання графіки  
def begin\_graphics(width=640, height=480, color=formatColor(0, 0, 0), title=None):  
  
 global \_root\_window, \_canvas, \_canvas\_x, \_canvas\_y, \_canvas\_xs, \_canvas\_ys, \_bg\_color  
  
 # Збереження параметрів розміру полотна  
 \_canvas\_xs, \_canvas\_ys = width - 1, height - 1  
 \_canvas\_x, \_canvas\_y = 0, \_canvas\_ys  
 \_bg\_color = color  
  
 # Створення кореневого вікна  
 \_root\_window = tkinter.Tk()  
 \_root\_window.protocol('WM\_DELETE\_WINDOW', \_destroy\_window)  
 \_root\_window.title(title or 'Graphics Window')  
 \_root\_window.resizable(0, 0)  
  
 # Створення об'єкта полотна  
 try:  
 \_canvas = tkinter.Canvas(\_root\_window, width=width, height=height)  
 \_canvas.pack()  
 draw\_background()  
 \_canvas.update()  
 except:  
 \_root\_window = None  
 raise  
  
 # Прив'язка івентів до натискань  
 \_root\_window.bind( "<KeyPress>", \_keypress )  
 \_root\_window.bind( "<KeyRelease>", \_keyrelease )  
 \_root\_window.bind( "<FocusIn>", \_clear\_keys )  
 \_root\_window.bind( "<FocusOut>", \_clear\_keys )  
 \_root\_window.bind( "<Button-1>", \_leftclick )  
 \_root\_window.bind( "<Button-2>", \_rightclick )  
 \_root\_window.bind( "<Button-3>", \_rightclick )  
 \_root\_window.bind( "<Control-Button-1>", \_ctrl\_leftclick)  
 \_clear\_keys()  
  
\_leftclick\_loc = None  
\_rightclick\_loc = None  
\_ctrl\_leftclick\_loc = None  
  
# Опрацювання натискання клавіш  
def \_leftclick(event):  
 global \_leftclick\_loc  
 \_leftclick\_loc = (event.x, event.y)  
  
def \_rightclick(event):  
 global \_rightclick\_loc  
 \_rightclick\_loc = (event.x, event.y)  
  
def \_ctrl\_leftclick(event):  
 global \_ctrl\_leftclick\_loc  
 \_ctrl\_leftclick\_loc = (event.x, event.y)  
  
# Відмальовка заднього фону  
def draw\_background():  
 corners = [(0,0), (0, \_canvas\_ys), (\_canvas\_xs, \_canvas\_ys), (\_canvas\_xs, 0)]  
 polygon(corners, \_bg\_color, fillColor=\_bg\_color, filled=True, smoothed=False)  
  
# Вихід із вікна  
def \_destroy\_window():  
 sys.exit(0)  
  
# Рендеринг полігонів  
def polygon(coords, outlineColor, fillColor=None, filled=1, smoothed=1, behind=0, width=1):  
 c = []  
 for coord in coords:  
 c.append(coord[0])  
 c.append(coord[1])  
 if fillColor == None: fillColor = outlineColor  
 if filled == 0: fillColor = ""  
 poly = \_canvas.create\_polygon(c, outline=outlineColor, fill=fillColor, smooth=smoothed, width=width)  
 if behind > 0:  
 \_canvas.tag\_lower(poly, behind) # Higher should be more visible  
 return poly  
  
  
  
# Рендеринг кіл  
def circle(pos, r, outlineColor, fillColor=None, endpoints=None, style='pieslice', width=2):  
 x, y = pos  
 x0, x1 = x - r - 1, x + r  
 y0, y1 = y - r - 1, y + r  
 if endpoints == None:  
 e = [0, 359]  
 else:  
 e = list(endpoints)  
 while e[0] > e[1]: e[1] = e[1] + 360  
  
 return \_canvas.create\_arc(x0, y0, x1, y1, outline=outlineColor, fill=fillColor or outlineColor,  
 extent=e[1] - e[0], start=e[0], style=style, width=width)  
  
# Метод для переміщення кіл  
def moveCircle(id, pos, r, endpoints=None):  
 global \_canvas\_x, \_canvas\_y  
  
 x, y = pos  
 x0, x1 = x - r - 1, x + r  
 y0, y1 = y - r - 1, y + r  
 if endpoints == None:  
 e = [0, 359]  
 else:  
 e = list(endpoints)  
 while e[0] > e[1]: e[1] = e[1] + 360  
  
 if os.path.isfile('flag'):  
 edit(id, ('extent', e[1] - e[0]))  
 else:  
 edit(id, ('start', e[0]), ('extent', e[1] - e[0]))  
 move\_to(id, x0, y0)  
  
# Зміна об'єкту на полотні  
def edit(id, \*args):  
 \_canvas.itemconfigure(id, \*\*dict(args))  
  
# Додавання тексту на полотно  
def text(pos, color, contents, font='Helvetica', size=12, style='normal', anchor="nw"):  
 global \_canvas\_x, \_canvas\_y  
 x, y = pos  
 font = (font, str(size), style)  
 return \_canvas.create\_text(x, y, fill=color, text=contents, font=font, anchor=anchor)  
  
# Зміна тексту на полотні  
def changeText(id, newText, font=None, size=12, style='normal'):  
 \_canvas.itemconfigure(id, text=newText)  
 if font != None:  
 \_canvas.itemconfigure(id, font=(font, '-%d' % size, style))  
  
  
# Додавання ліній на полотно  
def line(here, there, color=formatColor(0, 0, 0), width=2):  
 x0, y0 = here[0], here[1]  
 x1, y1 = there[0], there[1]  
 return \_canvas.create\_line(x0, y0, x1, y1, fill=color, width=width)  
  
  
# Ці словники зберігають неопрацьовані відпускаяння клавіш. Ми зробили затримку відпускання клавіш до  
# до одного виклику методу keys\_pressed() щоб обійти проблему з автоповторенням  
\_keysdown = {}  
\_keyswaiting = {}  
\_got\_release = None  
  
# Опрацювання натискання клавіші  
def \_keypress(event):  
 global \_got\_release  
 \_keysdown[event.keysym] = 1  
 \_keyswaiting[event.keysym] = 1  
 \_got\_release = None  
  
# Опрацювання відпускання клавіші  
def \_keyrelease(event):  
 global \_got\_release  
 try:  
 del \_keysdown[event.keysym]  
 except:  
 pass  
 \_got\_release = 1  
  
# Очистити неопрацьовані клавіші  
def \_clear\_keys():  
 global \_keysdown, \_got\_release, \_keyswaiting  
 \_keysdown = {}  
 \_keyswaiting = {}  
 \_got\_release = None  
  
# Опрацювання натиснутих клавіш  
def keys\_pressed(d\_o\_e=lambda arg: \_root\_window.dooneevent(arg),  
 d\_w=tkinter.\_tkinter.DONT\_WAIT):  
 d\_o\_e(d\_w)  
 if \_got\_release:  
 d\_o\_e(d\_w)  
 return \_keysdown.keys()  
  
# Повернення клавіш, які очікують  
def keys\_waiting():  
 global \_keyswaiting  
 keys = \_keyswaiting.keys()  
 \_keyswaiting = {}  
 return keys  
  
# Block for a list of keys...  
  
# Затримка для відпускання ключів  
def wait\_for\_keys():  
 keys = []  
 while keys == []:  
 keys = keys\_pressed()  
 sleep(0.05)  
 return keys  
  
# Метод для прибирання об'єкту із екрану  
def remove\_from\_screen(x,  
 d\_o\_e=lambda arg: \_root\_window.dooneevent(arg),  
 d\_w=tkinter.\_tkinter.DONT\_WAIT):  
 \_canvas.delete(x)  
 d\_o\_e(d\_w)  
  
  
# Переміщення об'єкту в інші координати  
def move\_to(object, x, y=None,  
 d\_o\_e=lambda arg: \_root\_window.dooneevent(arg),  
 d\_w=tkinter.\_tkinter.DONT\_WAIT):  
 if y is None:  
 try: x, y = x  
 except: raise 'incomprehensible coordinates'  
  
 horiz = True  
 newCoords = []  
 current\_x, current\_y = \_canvas.coords(object)[0:2] # first point  
 for coord in \_canvas.coords(object):  
 if horiz:  
 inc = x - current\_x  
 else:  
 inc = y - current\_y  
 horiz = not horiz  
  
 newCoords.append(coord + inc)  
  
 \_canvas.coords(object, \*newCoords)  
 d\_o\_e(d\_w)  
  
# Переміщення об'єкта на деяку відстань  
def move\_by(object, x, y=None,  
 d\_o\_e=lambda arg: \_root\_window.dooneevent(arg),  
 d\_w=tkinter.\_tkinter.DONT\_WAIT, lift=False):  
 if y is None:  
 try: x, y = x  
 except: raise Exception('incomprehensible coordinates')  
  
 horiz = True  
 newCoords = []  
 for coord in \_canvas.coords(object):  
 if horiz:  
 inc = x  
 else:  
 inc = y  
 horiz = not horiz  
  
 newCoords.append(coord + inc)  
  
 \_canvas.coords(object, \*newCoords)  
 d\_o\_e(d\_w)  
 if lift:  
 \_canvas.tag\_raise(object)  
  
# Форми привидів  
ghost\_shape = [  
 (0, - 0.5),  
 (0.25, - 0.75),  
 (0.5, - 0.5),  
 (0.75, - 0.75),  
 (0.75, 0.5),  
 (0.5, 0.75),  
 (- 0.5, 0.75),  
 (- 0.75, 0.5),  
 (- 0.75, - 0.75),  
 (- 0.5, - 0.5),  
 (- 0.25, - 0.75)  
 ]  
  
# Вхід в програму  
if \_\_name\_\_ == '\_\_main\_\_':  
 begin\_graphics()  
 ghost\_shape = [(x \* 10 + 20, y \* 10 + 20) for x, y in ghost\_shape]  
 g = polygon(ghost\_shape, formatColor(1, 1, 1))  
 move\_to(g, (50, 50))  
 circle((150, 150), 20, formatColor(0.7, 0.3, 0.0), endpoints=[15, - 15])  
 sleep(2)

keyboardAgent.py

from game import Agent  
from game import Directions  
import random  
  
  
# Агент, який контролюється із клавіатури  
class KeyboardAgent(Agent):  
 WEST\_KEY = 'a'  
 EAST\_KEY = 'd'  
 NORTH\_KEY = 'w'  
 SOUTH\_KEY = 's'  
 STOP\_KEY = 'q'  
  
 def \_\_init\_\_(self, index=0):  
  
 super().\_\_init\_\_(index)  
 self.lastMove = Directions.STOP  
 self.index = index  
 self.keys = []  
  
 # Отримати дію від агента  
 def getAction(self, state):  
 from graphicsUtils import keys\_waiting  
 from graphicsUtils import keys\_pressed  
 keys = list(keys\_waiting()) + list(keys\_pressed())  
 if keys != []:  
 self.keys = keys  
  
 legal = state.getLegalActions(self.index)  
 move = self.getMove(legal)  
  
 if move == Directions.STOP:  
 if self.lastMove in legal:  
 move = self.lastMove  
  
 if (self.STOP\_KEY in self.keys) and Directions.STOP in legal: move = Directions.STOP  
  
 if move not in legal:  
 move = random.choice(legal)  
  
 self.lastMove = move  
 return move  
  
 # Опрацювання команди на переміщення  
 def getMove(self, legal):  
 move = Directions.STOP  
 if (self.WEST\_KEY in self.keys or 'Left' in self.keys) and Directions.WEST in legal: move = Directions.WEST  
 if (self.EAST\_KEY in self.keys or 'Right' in self.keys) and Directions.EAST in legal: move = Directions.EAST  
 if (self.NORTH\_KEY in self.keys or 'Up' in self.keys) and Directions.NORTH in legal: move = Directions.NORTH  
 if (self.SOUTH\_KEY in self.keys or 'Down' in self.keys) and Directions.SOUTH in legal: move = Directions.SOUTH  
 return move

**layout.py**

from game import Grid  
import os  
  
  
# Керує статичною інформацією про поле гри  
class Layout:  
  
 def \_\_init\_\_(self, layoutText):  
 self.width = len(layoutText[0])  
 self.height = len(layoutText)  
 self.walls = Grid(self.width, self.height, False)  
 self.food = Grid(self.width, self.height, False)  
 self.capsules = []  
 self.agentPositions = []  
 self.numGhosts = 0  
 self.processLayoutText(layoutText)  
 self.layoutText = layoutText  
  
 # Отримати кількість привидів  
 def getNumGhosts(self):  
 return self.numGhosts  
  
 def deepCopy(self):  
 return Layout(self.layoutText[:])  
  
 # Опрацювання символів із написаних полів для гри  
 # % - Стіна  
 # . - Їжа  
 # o - Капсула  
 # G - Привид  
 # P - Pac-man  
 def processLayoutText(self, layoutText):  
 maxY = self.height - 1  
 for y in range(self.height):  
 for x in range(self.width):  
 layoutChar = layoutText[maxY - y][x]  
 self.processLayoutChar(x, y, layoutChar)  
 self.agentPositions.sort()  
 self.agentPositions = [(i == 0, pos) for i, pos in self.agentPositions]  
  
 # Опрацювання одного символу  
 def processLayoutChar(self, x, y, layoutChar):  
 if layoutChar == '%':  
 self.walls[x][y] = True  
 elif layoutChar == '.':  
 self.food[x][y] = True  
 elif layoutChar == 'o':  
 self.capsules.append((x, y))  
 elif layoutChar == 'P':  
 self.agentPositions.append((0, (x, y)))  
 elif layoutChar in ['G']:  
 self.agentPositions.append((1, (x, y)))  
 self.numGhosts += 1  
 elif layoutChar in ['1', '2', '3', '4']:  
 self.agentPositions.append((int(layoutChar), (x, y)))  
 self.numGhosts += 1  
  
  
# Завантаження поля  
def getLayout(name, back=2):  
 if name.endswith('.lay'):  
 layout = tryToLoad('layouts/' + name)  
 if layout is None:  
 layout = tryToLoad(name)  
 else:  
 layout = tryToLoad('layouts/' + name + '.lay')  
 if layout is None:  
 layout = tryToLoad(name + '.lay')  
 if layout is None and back >= 0:  
 curdir = os.path.abspath('.')  
 os.chdir('..')  
 layout = getLayout(name, back - 1)  
 os.chdir(curdir)  
 return layout  
  
  
# Завантаження поля із заданою назвою  
def tryToLoad(fullname):  
 if not os.path.exists(fullname):  
 return None  
 f = open(fullname)  
 try:  
 return Layout([line.strip() for line in f])  
 finally:  
 f.close()

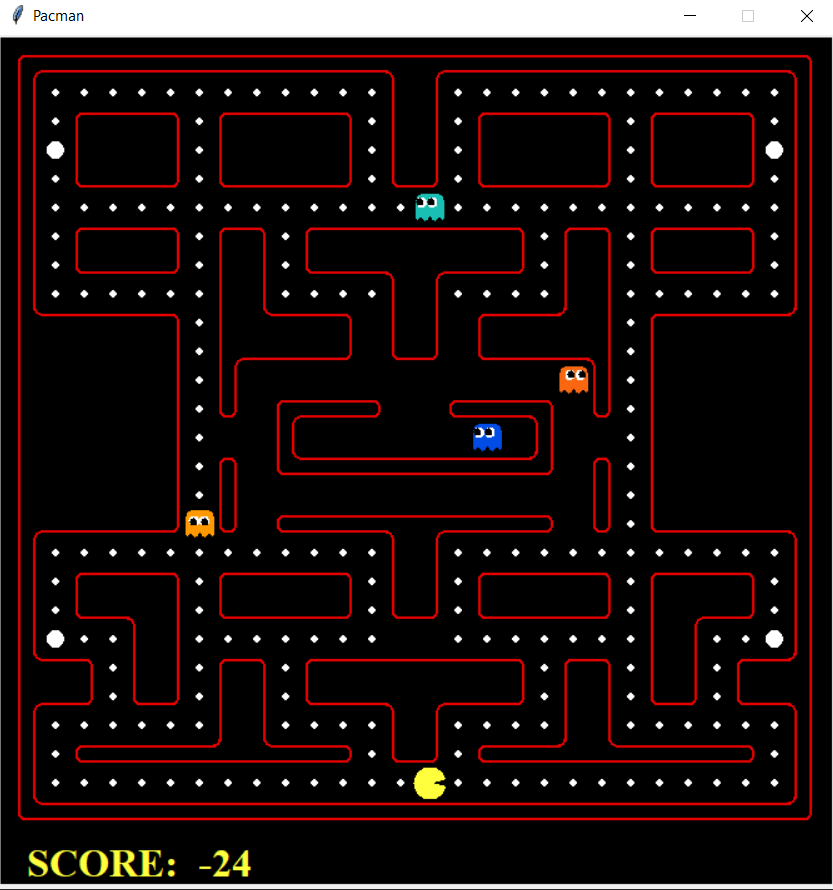
**pacman.py**

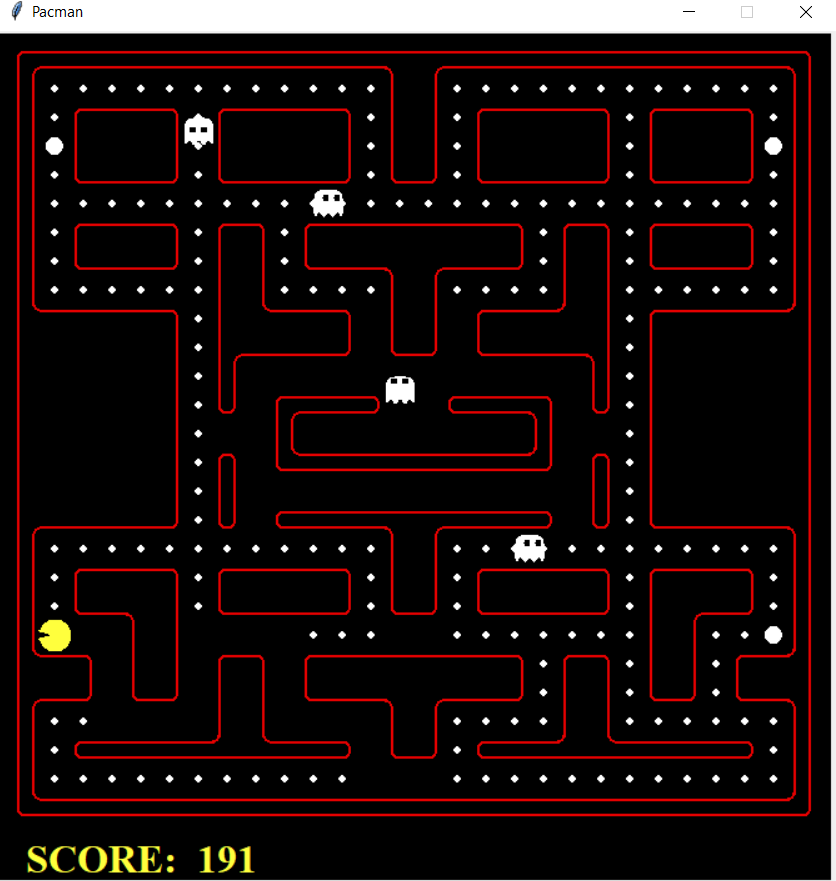
import os  
import random  
import sys  
  
import layout  
from game import Actions  
from game import Directions  
from game import Game  
from game import GameStateData  
from util import manhattanDistance  
from util import nearestPoint  
from datetime import datetime  
  
  
# Визначає повний стан гри, включаючи їжу, капсули, конфігурації агентів та зміни очок.  
class GameState:  
 explored = set()  
  
 # Повертає можливі дії щодо зазначеного агента.  
 def getLegalActions(self, agentIndex=0):  
  
 if self.isWin() or self.isLose(): return []  
  
 if agentIndex == 0: # Pacman is moving  
 return PacmanRules.getLegalActions(self)  
 else:  
 return GhostRules.getLegalActions(self, agentIndex)  
  
 # Генерує наступний стан після того, як зазначений агент виконає дію.  
 def generateSuccessor(self, agentIndex, action):  
  
 if self.isWin() or self.isLose():  
 raise Exception('Can\'t generate a successor of a terminal state.')  
  
 state = GameState(self)  
  
 if agentIndex == 0: # Pacman is moving  
 state.data.\_eaten = [False for i in range(state.getNumAgents())]  
 PacmanRules.applyAction(state, action)  
 else: # A ghost is moving  
 GhostRules.applyAction(state, action, agentIndex)  
  
 if agentIndex == 0:  
 state.data.scoreChange += -TIME\_PENALTY # Penalty for waiting around  
 else:  
 GhostRules.decrementTimer(state.data.agentStates[agentIndex])  
  
 GhostRules.checkDeath(state, agentIndex)  
  
 state.data.\_agentMoved = agentIndex  
 state.data.score += state.data.scoreChange  
 GameState.explored.add(self)  
 GameState.explored.add(state)  
 return state  
  
 # Повернути можливі дії для Pac-man  
 def getLegalPacmanActions(self):  
 return self.getLegalActions(0)  
  
 # Генерує наступний стан, після того, як Pac-man зробить хід  
 def generatePacmanSuccessor(self, action):  
 return self.generateSuccessor(0, action)  
  
 # Повернути статус Pac-man  
 def getPacmanState(self):  
 return self.data.agentStates[0].copy()  
  
 # Повернути позицію Pac-man  
 def getPacmanPosition(self):  
 return self.data.agentStates[0].getPosition()  
  
 # Повернути статус привида  
 def getGhostState(self, agentIndex):  
 if agentIndex == 0 or agentIndex >= self.getNumAgents():  
 raise Exception("Invalid index passed to getGhostState")  
 return self.data.agentStates[agentIndex]  
  
 # Повернути загальну кількість всіх агентів  
 def getNumAgents(self):  
 return len(self.data.agentStates)  
  
 # Повернути місцеположення залишених капсул  
 def getCapsules(self):  
 return self.data.capsules  
  
 # Повернути кількість їжі  
 def getNumFood(self):  
 return self.data.food.count()  
  
 # Повернення статусу гри (перемога/поразка)  
 def isLose(self):  
 return self.data.\_lose  
  
 def isWin(self):  
 return self.data.\_win  
  
 def \_\_init\_\_(self, prevState=None):  
 if prevState is not None: # Initial state  
 self.data = GameStateData(prevState.data)  
 else:  
 self.data = GameStateData()  
  
 def deepCopy(self):  
 state = GameState(self)  
 state.data = self.data.deepCopy()  
 return state  
  
 # Створює початковий стан гри з масиву макета  
 def initialize(self, layout, numGhostAgents=1000):  
 *"""  
 Creates an initial game state from a layout array (see layout.py).  
 """* self.data.initialize(layout, numGhostAgents)  
  
  
# Час наляканості привидів  
SCARED\_TIME = 40  
# Наскільки близько мають знаходитися привиди для того, щоб вбити Pac-man  
COLLISION\_TOLERANCE = 0.7  
# Кількість очок, які губляться із кожним циклом гри  
TIME\_PENALTY = 1  
  
  
# Правила гри керують потоком управління грою, вирішуючи, коли і як гра починається і закінчується.  
class ClassicGameRules:  
  
 def \_\_init\_\_(self, timeout=30):  
 self.timeout = timeout  
  
 # Ініціалізація нової гри  
 def newGame(self, layout, pacmanAgent, ghostAgents, display, catchExceptions=False):  
 agents = [pacmanAgent] + ghostAgents[:layout.getNumGhosts()]  
 initState = GameState()  
 initState.initialize(layout, len(ghostAgents))  
 game = Game(agents, display, self, catchExceptions=catchExceptions)  
 game.state = initState  
 self.initialState = initState.deepCopy()  
 return game  
  
 # Перевіряє, чи пора закінчувати гру.  
 def process(self, state, game):  
 if state.isWin():  
 self.win(state, game)  
 if state.isLose():  
 self.lose(state, game)  
  
 def win(self, state, game):  
 print("Pacman emerges victorious! Score: %d" % state.data.score)  
 game.gameOver = True  
  
 def lose(self, state, game):  
 print("Pacman died! Score: %d" % state.data.score)  
 game.gameOver = True  
  
 # Отримати прогрес гри  
 def getProgress(self, game):  
 return float(game.state.getNumFood()) / self.initialState.getNumFood()  
  
  
# Ці функції визначають, як Pac-man взаємодіє з навколишнім середовищем за класичними правилами гри.  
class PacmanRules:  
 PACMAN\_SPEED = 1  
  
 # Повертає список можливих дій.  
 def getLegalActions(state):  
 return Actions.getPossibleActions(state.getPacmanState().configuration, state.data.layout.walls)  
  
 getLegalActions = staticmethod(getLegalActions)  
  
 # Редагує стан, щоб відображати результати дії.  
 def applyAction(state, action):  
 legal = PacmanRules.getLegalActions(state)  
 if action not in legal:  
 raise Exception("Illegal action " + str(action))  
  
 pacmanState = state.data.agentStates[0]  
  
 vector = Actions.directionToVector(action, PacmanRules.PACMAN\_SPEED)  
 pacmanState.configuration = pacmanState.configuration.generateSuccessor(vector)  
  
 next = pacmanState.configuration.getPosition()  
 nearest = nearestPoint(next)  
 if manhattanDistance(nearest, next) <= 0.5:  
 PacmanRules.consume(nearest, state)  
  
 applyAction = staticmethod(applyAction)  
  
 # Опрацювання вживання їжі/капсул Pac-man  
 def consume(position, state):  
 x, y = position  
 if state.data.food[x][y]:  
 state.data.scoreChange += 10  
 state.data.food = state.data.food.copy()  
 state.data.food[x][y] = False  
 state.data.\_foodEaten = position  
 numFood = state.getNumFood()  
 if numFood == 0 and not state.data.\_lose:  
 state.data.scoreChange += 500  
 state.data.\_win = True  
 if position in state.getCapsules():  
 state.data.capsules.remove(position)  
 state.data.\_capsuleEaten = position  
 for index in range(1, len(state.data.agentStates)):  
 state.data.agentStates[index].scaredTimer = SCARED\_TIME  
  
 consume = staticmethod(consume)  
  
  
# Ці функції визначають, як привиди взаємодіють із своїм оточенням.  
class GhostRules:  
 GHOST\_SPEED = 1.0  
  
 # Дії, які можуть робити привиди  
 def getLegalActions(state, ghostIndex):  
 conf = state.getGhostState(ghostIndex).configuration  
 possibleActions = Actions.getPossibleActions(conf, state.data.layout.walls)  
 reverse = Actions.reverseDirection(conf.direction)  
 if Directions.STOP in possibleActions:  
 possibleActions.remove(Directions.STOP)  
 if reverse in possibleActions and len(possibleActions) > 1:  
 possibleActions.remove(reverse)  
 return possibleActions  
  
 getLegalActions = staticmethod(getLegalActions)  
  
 # Редагує стан, щоб відображати результати дії.  
 def applyAction(state, action, ghostIndex):  
  
 legal = GhostRules.getLegalActions(state, ghostIndex)  
 if action not in legal:  
 raise Exception("Illegal ghost action " + str(action))  
  
 ghostState = state.data.agentStates[ghostIndex]  
 speed = GhostRules.GHOST\_SPEED  
 if ghostState.scaredTimer > 0: speed /= 2.0  
 vector = Actions.directionToVector(action, speed)  
 ghostState.configuration = ghostState.configuration.generateSuccessor(vector)  
  
 applyAction = staticmethod(applyAction)  
  
 # Зменшення часу наляканості  
 def decrementTimer(ghostState):  
 timer = ghostState.scaredTimer  
 if timer == 1:  
 ghostState.configuration.pos = nearestPoint(ghostState.configuration.pos)  
 ghostState.scaredTimer = max(0, timer - 1)  
  
 decrementTimer = staticmethod(decrementTimer)  
  
 # Перевірка того, чи вмер привид  
 def checkDeath(state, agentIndex):  
 pacmanPosition = state.getPacmanPosition()  
 if agentIndex == 0:  
 for index in range(1, len(state.data.agentStates)):  
 ghostState = state.data.agentStates[index]  
 ghostPosition = ghostState.configuration.getPosition()  
 if GhostRules.canKill(pacmanPosition, ghostPosition):  
 GhostRules.collide(state, ghostState, index)  
 else:  
 ghostState = state.data.agentStates[agentIndex]  
 ghostPosition = ghostState.configuration.getPosition()  
 if GhostRules.canKill(pacmanPosition, ghostPosition):  
 GhostRules.collide(state, ghostState, agentIndex)  
  
 checkDeath = staticmethod(checkDeath)  
  
 # Колізія  
 def collide(state, ghostState, agentIndex):  
 if ghostState.scaredTimer > 0:  
 state.data.scoreChange += 200  
 GhostRules.placeGhost(ghostState)  
 ghostState.scaredTimer = 0  
 state.data.\_eaten[agentIndex] = True  
 else:  
 if not state.data.\_win:  
 state.data.scoreChange -= 500  
 state.data.\_lose = True  
  
 collide = staticmethod(collide)  
  
 # Перевірка того, чи може привид вбити Pac-man  
 def canKill(pacmanPosition, ghostPosition):  
 return manhattanDistance(ghostPosition, pacmanPosition) <= COLLISION\_TOLERANCE  
  
 canKill = staticmethod(canKill)  
  
 # Розмістити привида  
 def placeGhost(ghostState):  
 ghostState.configuration = ghostState.start  
  
 placeGhost = staticmethod(placeGhost)  
  
  
# Позначення базової строки  
def default(str):  
 return str + ' [Default: %default]'  
  
  
# Парсинг аргументів консолі  
def parseAgentArgs(str):  
 if str == None: return {}  
 pieces = str.split(',')  
 opts = {}  
 for p in pieces:  
 if '=' in p:  
 key, val = p.split('=')  
 else:  
 key, val = p, 1  
 opts[key] = val  
 return opts  
  
  
# Оброблення команди, що використовується для запуску Pac-man з командного рядка.  
def readCommand(argv):  
 from optparse import OptionParser  
 usageStr = """  
 USAGE: python pacman.py <options>  
 EXAMPLES: (1) python pacman.py  
 - starts an interactive game  
 (2) python pacman.py --layout smallClassic --zoom 2  
 OR python pacman.py -l smallClassic -z 2  
 - starts an interactive game on a smaller board, zoomed in  
 """  
 parser = OptionParser(usageStr)  
  
 parser.add\_option('-l', '--layout', dest='layout',  
 help=default('the LAYOUT\_FILE from which to load the map layout'),  
 metavar='LAYOUT\_FILE', default='originalClassic')  
 parser.add\_option('-p', '--pacman', dest='pacman',  
 help=default('the agent TYPE in the pacmanAgents module to use'),  
 metavar='TYPE', default='KeyboardAgent')  
 parser.add\_option('-t', '--textGraphics', action='store\_true', dest='textGraphics',  
 help='Display output as text only', default=False)  
 parser.add\_option('-q', '--quietTextGraphics', action='store\_true', dest='quietGraphics',  
 help='Generate minimal output and no graphics', default=False)  
 parser.add\_option('-g', '--ghosts', dest='ghost',  
 help=default('the ghost agent TYPE in the ghostAgents module to use'),  
 metavar='TYPE', default='RandomGhost')  
 parser.add\_option('-k', '--numghosts', type='int', dest='numGhosts',  
 help=default('The maximum number of ghosts to use'), default=4)  
 parser.add\_option('-z', '--zoom', type='float', dest='zoom',  
 help=default('Zoom the size of the graphics window'), default=1.0)  
 parser.add\_option('-f', '--fixRandomSeed', action='store\_true', dest='fixRandomSeed',  
 help='Fixes the random seed to always play the same game', default=False)  
 parser.add\_option('-r', '--recordActions', action='store\_true', dest='record',  
 help='Writes game histories to a file (named by the time they were played)', default=False)  
 parser.add\_option('--replay', dest='gameToReplay',  
 help='A recorded game file (pickle) to replay', default=None)  
 parser.add\_option('-a', '--agentArgs', dest='agentArgs',  
 help='Comma separated values sent to agent. e.g. "opt1=val1,opt2,opt3=val3"')  
 parser.add\_option('-x', '--numTraining', dest='numTraining', type='int',  
 help=default('How many episodes are training (suppresses output)'), default=0)  
 parser.add\_option('--frameTime', dest='frameTime', type='float',  
 help=default('Time to delay between frames; <0 means keyboard'), default=0.1)  
 parser.add\_option('-c', '--catchExceptions', action='store\_true', dest='catchExceptions',  
 help='Turns on exception handling and timeouts during games', default=False)  
 parser.add\_option('--timeout', dest='timeout', type='int',  
 help=default('Maximum length of time an agent can spend computing in a single game'), default=30)  
  
 options, otherjunk = parser.parse\_args(argv)  
 if len(otherjunk) != 0:  
 raise Exception('Command line input not understood: ' + str(otherjunk))  
 args = dict()  
  
 # Fix the random seed  
 if options.fixRandomSeed: random.seed('cs188')  
  
 # Choose a layout  
 args['layout'] = layout.getLayout(options.layout)  
 # Choose a Pacman agent  
 if args['layout'] is not None:  
 pacmanType = loadAgent(options.pacman)  
 agentOpts = parseAgentArgs(options.agentArgs)  
 pacman = pacmanType(\*\*agentOpts) # Instantiate Pacman with agentArgs  
 args['pacman'] = pacman  
  
 # Choose a ghost agent  
 ghostType = loadAgent(options.ghost)  
 args['ghosts'] = [ghostType(i + 1) for i in range(options.numGhosts)]  
  
 # Choose a display format  
 import graphicsDisplay  
 args['display'] = graphicsDisplay.PacmanGraphics(options.zoom, frameTime=options.frameTime)  
  
 args['catchExceptions'] = options.catchExceptions  
 args['timeout'] = options.timeout  
  
 return args  
  
 raise Exception("The layout " + options.layout + " cannot be found")  
  
  
# Завантаження агента  
def loadAgent(pacman):  
 pythonPathStr = os.path.expandvars("$PYTHONPATH")  
 if pythonPathStr.find(';') == -1:  
 pythonPathDirs = pythonPathStr.split(':')  
 else:  
 pythonPathDirs = pythonPathStr.split(';')  
 pythonPathDirs.append('.')  
  
 for moduleDir in pythonPathDirs:  
 if not os.path.isdir(moduleDir):  
 continue  
 moduleNames = [f for f in os.listdir(moduleDir) if f.endswith('gents.py')]  
 for modulename in moduleNames:  
 try:  
 module = \_\_import\_\_(modulename[:-3])  
 except ImportError:  
 continue  
 if pacman in dir(module):  
 return getattr(module, pacman)  
 raise Exception('The agent ' + pacman + ' is not specified in any \*Agents.py.')  
  
  
# Запуск ігор  
def runGames(layout, pacman, ghosts, display, catchExceptions=False, timeout=30):  
 import \_\_main\_\_  
 \_\_main\_\_.\_\_dict\_\_['\_display'] = display  
 rules = ClassicGameRules(timeout)  
 game = rules.newGame(layout, pacman, ghosts, display, catchExceptions)  
 startTime = datetime.now()  
 game.run()  
 endTime = datetime.now()  
 print('Game time:', endTime - startTime)  
 return game  
  
  
# Вхідна точка в програму  
if \_\_name\_\_ == '\_\_main\_\_':  
 args = readCommand(sys.argv[1:]) # Get game components based on input  
 runGames(\*\*args)

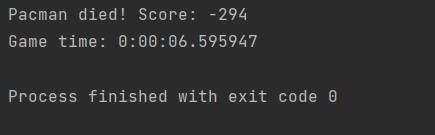
**util.py**

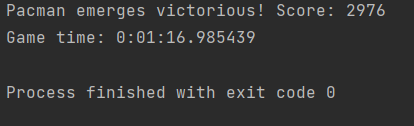
import sys  
import inspect  
import random  
  
  
# Повертає Мангеттенську відстань між двома точками  
def manhattanDistance( xy1, xy2 ):  
 return abs( xy1[0] - xy2[0] ) + abs( xy1[1] - xy2[1] )  
  
  
class Counter(dict):  
  
 # Повертає суму підрахунків для всіх ключів.  
 def totalCount(self):  
  
 return sum(self.values())  
  
 # Редагує лічильник таким чином, що загальна кількість усіх ключів дорівнює 1.  
 def normalize(self):  
 total = float(self.totalCount())  
 if total == 0:  
 return  
 for key in self.keys():  
 self[key] = self[key] / total  
  
  
# Допоміжна функція, якщо метод не імпелментовано  
def raiseNotDefined():  
 fileName = inspect.stack()[1][1]  
 line = inspect.stack()[1][2]  
 method = inspect.stack()[1][3]  
  
 print("\*\*\* Method not implemented: %s at line %s of %s" % (method, line, fileName))  
 sys.exit(1)  
  
  
# Нормалізація векторів або лічильників шляхом ділення кожного значення на суму всіх значень  
def normalize(vectorOrCounter):  
 normalizedCounter = Counter()  
 if type(vectorOrCounter) == type(normalizedCounter):  
 counter = vectorOrCounter  
 total = float(counter.totalCount())  
 if total == 0:  
 return counter  
 for key in counter.keys():  
 value = counter[key]  
 normalizedCounter[key] = value / total  
 return normalizedCounter  
 else:  
 vector = vectorOrCounter  
 s = float(sum(vector))  
 if s == 0:  
 return vector  
 return [el / s for el in vector]  
  
  
# Допоміжний метод для вибіркового розподілу  
def sample(distribution, values=None):  
 if type(distribution) == Counter:  
 items = sorted(distribution.items())  
 distribution = [i[1] for i in items]  
 values = [i[0] for i in items]  
 if sum(distribution) != 1:  
 distribution = normalize(distribution)  
 choice = random.random()  
 i, total = 0, distribution[0]  
 while choice > total:  
 i += 1  
 total += distribution[i]  
 return values[i]  
  
  
# Вибір елементу із розподілу  
def chooseFromDistribution(distribution):  
 if type(distribution) == dict or type(distribution) == Counter:  
 return sample(distribution)  
 r = random.random()  
 base = 0.0  
 for prob, element in distribution:  
 base += prob  
 if r <= base:  
 return element  
  
  
# Знаходить найближчу точку сітки до позиції (дискретно)  
def nearestPoint(pos):  
 (current\_row, current\_col) = pos  
  
 grid\_row = int(current\_row + 0.5)  
 grid\_col = int(current\_col + 0.5)  
 return grid\_row, grid\_col

# **Скріншоти роботи програмного застосунку**









Результатом даної роботи стала гра Pac-man, яка буде слугувати каркасом для майбутній лабораторних робіт.