Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Лабораторна робота 1

з дисципліни “Інтелектуальні системи”

“Пакмен”

Звіт

студента 4 курсу групи ТТП-42

Ходакова Максима Олеговича

Київ - 2024

**Постановка задачі**

Вимоги:

* Привиди мають підкорятись наперед запрограмованим протоколам. Сам пакмен же керується гравцем або заданою логікою;
* Код можна реалізовувати будь-якою мовою, а також дозволяється використання вже створеної візуалізації;
* Водночас має бути достатнє розуміння коду для внесення змін «на ходу»;
* Має бути реалізований та використаний принаймні один алгоритм пошуку;
* Генерація лабіринту з наростанням складності.

**Загальні параметри**

Вгорі програми можна встановити параметри виконання:

* WScreen=1280 – ширина вікна у пікселях
* HScreen=720 – висота вікна у пікселях
* WMaze=20 – ширина лабіринту у клітинках
* HMaze=20 – висота лабіринту у клітинках
* Difficulty=1 – складність (>=0)
* sizekoef=30 – розмір клітинки у пікселях(для масштабування)
* generation="WS" – метод генерації лабіринту(WS чи DFS)
* search="G" – метод пошуку(G чи A\*)
* showpath=True – чи вказувати шлях до монетки
* ghostnumber=6 – кількість привидів
* walldensity=0.3 – частка стін при генерації за WS

Гра реалізована у pygame. Для її роботи треба у Python встановити цей модуль.

**Генерація лабіринту**

Генерація лабіринту відбувається на початку першого рівня та після успішного проходження на наступний з наростанням складності

Програма генерує лабіринт двома методами: пошуком вглибину та почерговою вставкою стін у перестановці(WallShuffle).

При використанні пошуку вглибину виконується власне цей пошук з випадковим вибором напрямку пошуку. Стіни встановлюються всюди, де не було прямого переходу при пошуку. При цьому множина клітинок фактично є деревом, і привиди гарантовано можуть загнати гравця у глухий кут.

Інший метод визначає множину усіх можливих стін, робить її перестановку, відкидає кожну стіну з ймовірністю 1-walldensity, а інакше встановлює її за умови, що її додавання не створює глухий кут чи нову компоненту зв’язності.

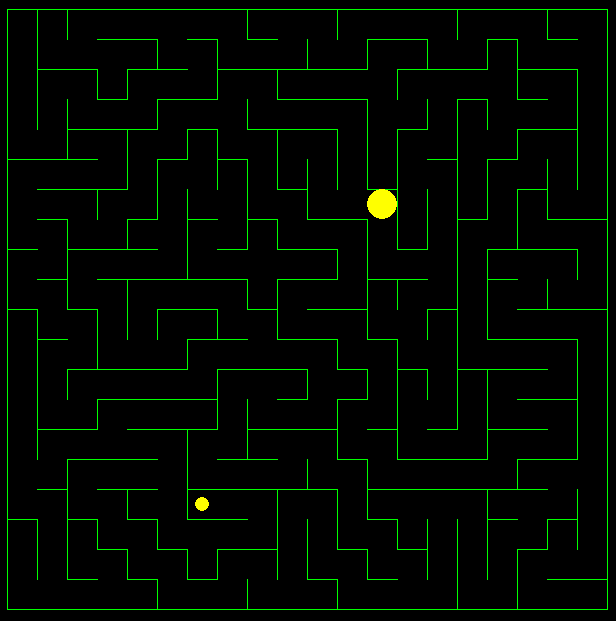


Рис.1 Приклад DFS-лабіринту

У прикладі для DFS дійсно бачимо велику кількість глухих кутів.

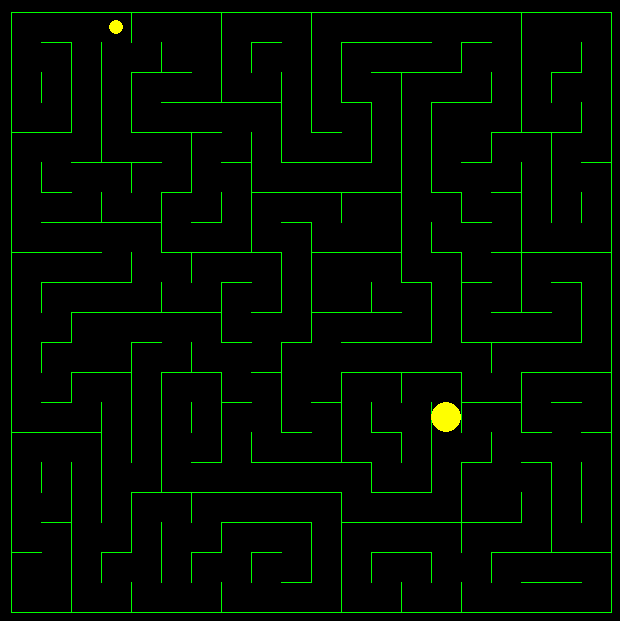
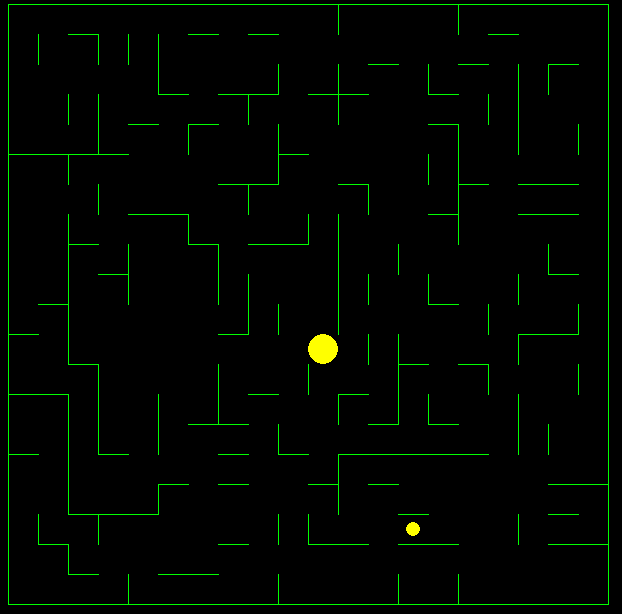


Рис.2 Приклад WS-лабіринту з walldensity 1

Рис.3 Приклад WS-лабіринту з walldensity 0.3

Два приклади для WS (walldensity рівні відповідно 1 і 0.3) показують, що залишається простір для обходу привидів та шляхи між клітинками можуть бути не єдиними. Використовувався у подальшому саме WS.

**Пошук у лабіринті**

Пошук виконується також двома методами: жадібним та A\*. У обох випадках використовується деяка міра зручності пошуку з клітинки, до якої пошук прийшов, але ще не виконав на ній свою ітерацію. Для наступної ітерації обирається клітинка з найменшою такою мірою. Як тільки потрібна клітинка знайдена, пошук повертає шлях до неї.

Жадібний метод використовує манхетенську відстань до шуканої клітинки в якості міри зручності, а метод A\* викорстовує суму жадібної міри та половини кількості клітинок на шляху до заданої. Таким чином жадібний метод, на відміну від звичайного пошуку вглибину, вибирає ті клітинки, які знаходяться ближче до шуканої, а метод A\* ще й додатково намагається час від часу виконати ітерації на достатньо близьких до початку клітинках, щоб «розширити» пошук вглибину багатьма варіантами напряму.

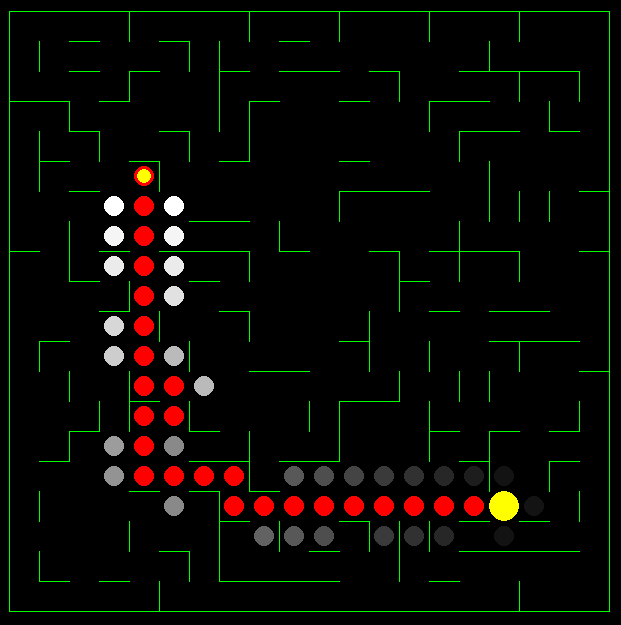
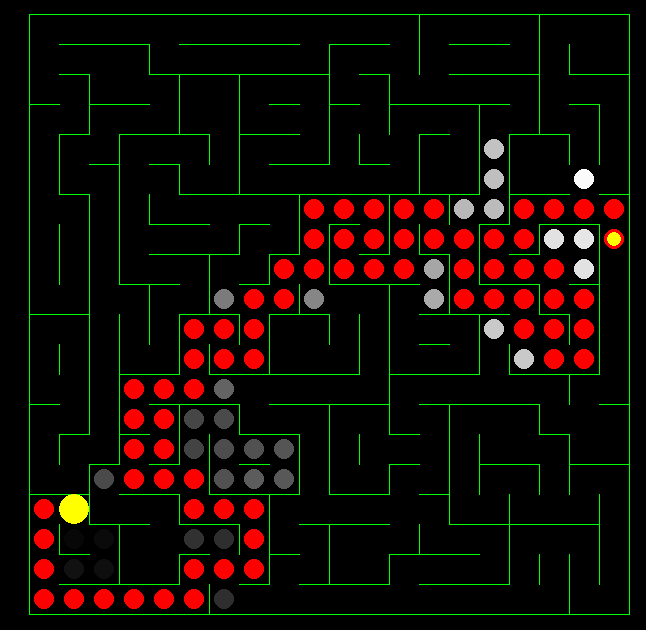


Рис.4 Пошук G у WS-лабіринті маленької густини

Рис.5 Пошук G у WS-лабіринті великої густини

Червоні круги – знайдений шлях від гравця до монети. Сірі – інші перебрані пошуком. Чим вони біліші, тим більша до них відстань. Для WS-лабіринту шляхів може бути багато, а тому методи не обов’язково дають найкоротший варіант. Проте жадібний метод за малої густини виконує пошук без перебору значної частки клітинок.

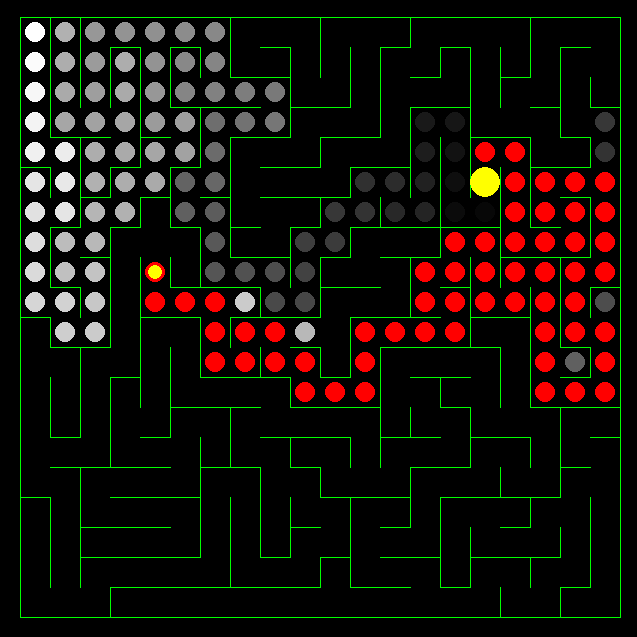
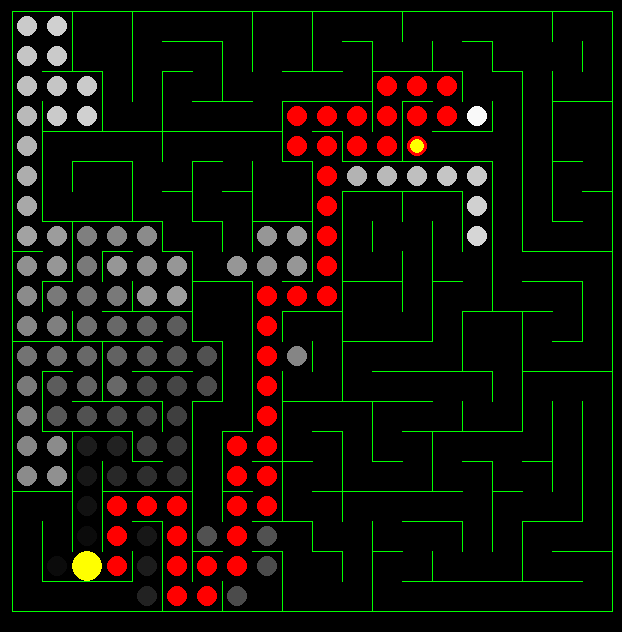


Рис.6 Пошук G у DFS-лабіринті

Рис.7 Пошук G у WS-лабіринті великої густини

Однак, звичайно, у обох генераціях можливі жадібні вибори тупикових шляхів.

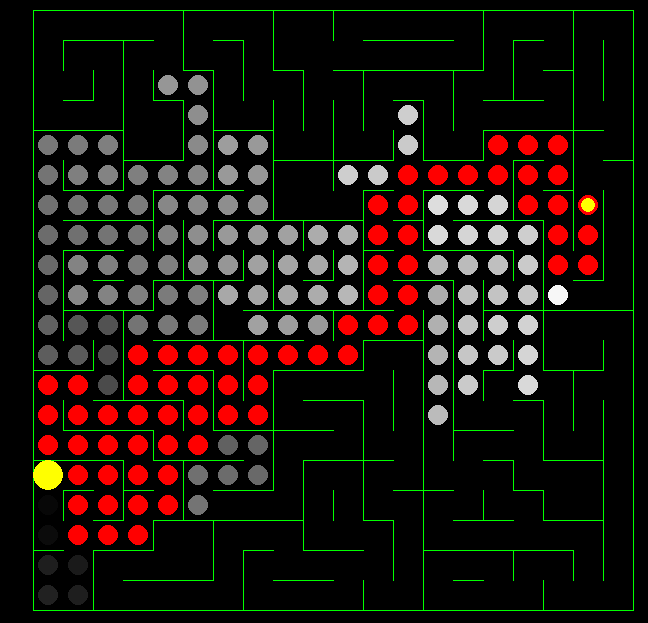
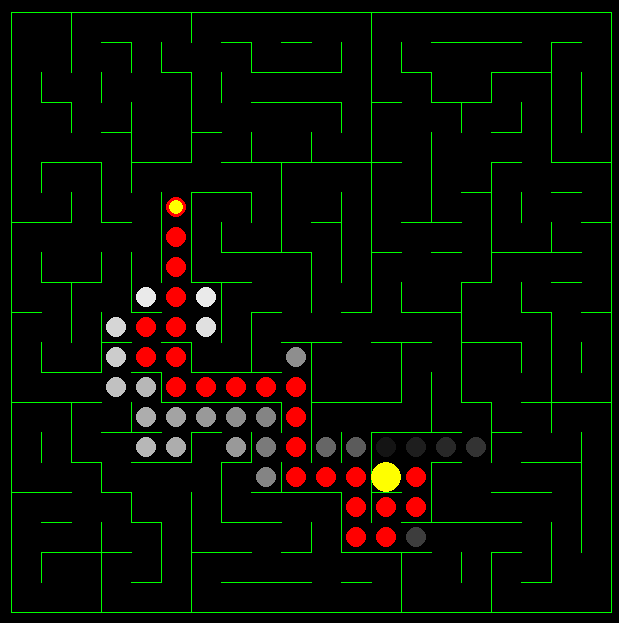


Рис.8 Пошук A\* у WS-лабіринті великої густини

Рис.9 Пошук A\* у WS-лабіринті малої густини

Пошук A\* подібний до жадібного, але намагається інколи перебирати і близькі до початку невигідні з точки зору жадібності клітини.

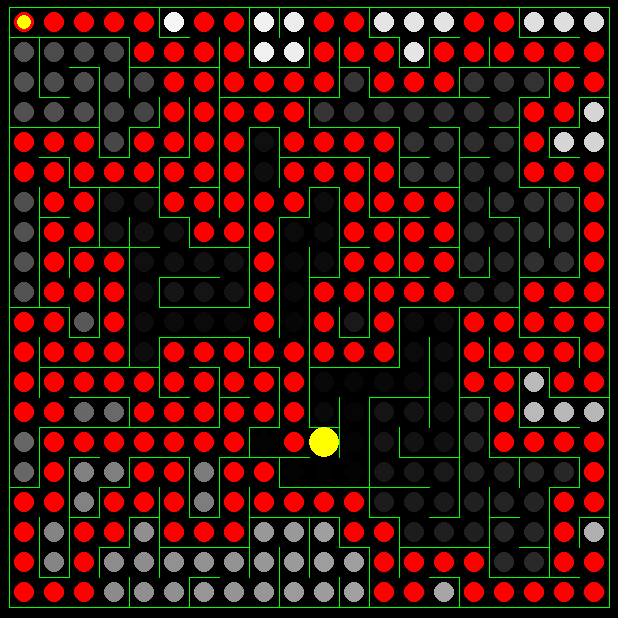
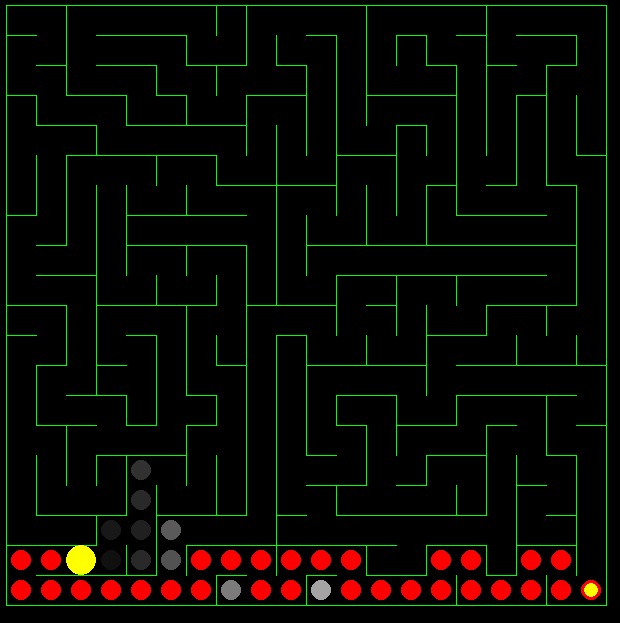


Рис.10 Пошук A\* у DFS-лабіринті

Рис.11 Пошук A\* у DFS-лабіринті

Останні два приклади показують, як у DFS-лабіринті шлях може залежати від випадку.

**Рух привидів та гравця**

Швидкість привидів на третину менша за швидкість гравця. Їх колір випадковий із синього спектру. Привиди можуть демонструвати як випадкову, так і розумну поведінку. Ймовірність вибору помилкової випадкової поведінки за кожного можливого вибору визнається формулою: probability = e^(-Difficulty / 3)

Розумна поведінка визначається наступним чином. Перший привид завжди за алгоритмом пошуку намагається досягти гравця. Другий та третій намагаються досягти позиції на колі з центром в позиції гравця зліва та справа від першого привиду. Таким чином, вони намагаються оточити гравця та напасти одночасно з першим привидом. Всі інші привиди намагаються зайняти позиції на очікуваному шляху від гравця до монетки. Це дозволяє створити перешкоди на шляху до монетки та забезпечити напружений момент із загальним нападом, коли гравець майже дістався монетки. Зрозуміло, кожен привид змінює тактику на звичайне переслідування, якщо гравець сильно до нього наблизився.

Загалом розумна поведінка привидів тримає гравця у змінній напрузі через комбіновану атаку переслідувачів та загрозу біля монетки, змушує його постійно оминати перешкоди. Ймовірність помилки дозволяє зменшити цю напругу та дозволити гравцю йти на ризик і самому допускати тактичні помилки.

Коли привид знаходиться на дуже малій відстані від гравця, гра завершується.

Гравець для руху нажимає відповідні стрілки клавіатури. Якщо напрям буде вказувати у стіну, то Пакмен зупиниться. При досяганні монети виводиться кількість набраних очок та створюється нова монетка у випадковій позиції так, щоб на прямій від гравця до монетки опинився «центр мас» привидів.

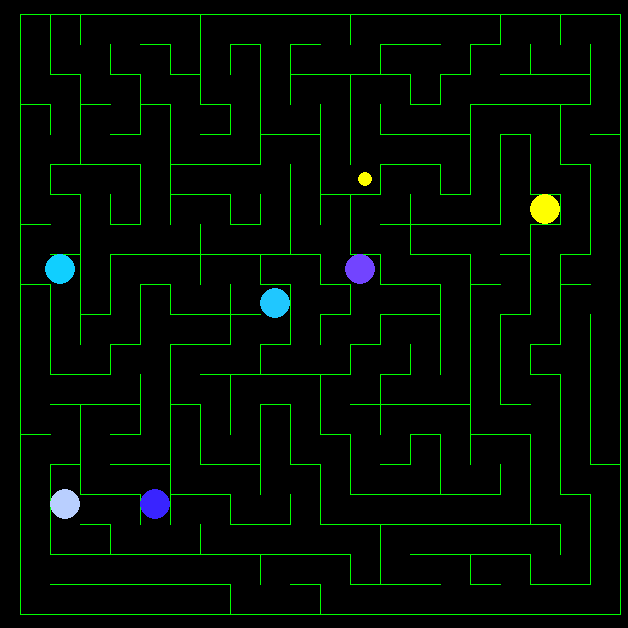


Рис.12 Лабіринт з привидами

Рис.13 Розподіл ролей у ігровій ситуації

**Додаток 1. Повний код разом з коментарями**

# Константи для розміру екрану та параметрів лабіринту  
*WScreen* = 1280 # Ширина екрану  
*HScreen* = 720 # Висота екрану  
*WMaze* = 20 # Ширина лабіринту (кількість клітин)  
*HMaze* = 20 # Висота лабіринту (кількість клітин)  
*Difficulty* = 0 # Рівень складності (>=0)  
*sizekoef* = 30 # Коефіцієнт розміру кожної клітинки  
*generation* = "WS" # Метод генерації лабіринту: "WS" або "DFS"  
*search* = "A\*" # Алгоритм пошуку шляху: "G" (жадібний) або "A\*"  
*showpath* = True # Відображати шлях чи ні  
*ghostnumber* = 4 # Кількість привидів на першому рівні  
*walldensity* = 0.3 # Щільність стін у лабіринті  
  
# Імпорт бібліотек для графіки та алгоритмів  
import pygame  
import random  
import math  
import queue  
import sys  
import heapq  
  
# Встановлюємо обмеження на кількість рекурсивних викликів  
sys.setrecursionlimit(*WMaze* \* *HMaze* + 10000)  
  
# Визначаємо початкові координати для центрування лабіринту на екрані  
*top* = (*HScreen* - *HMaze* \* *sizekoef*) // 2  
*left* = (*WScreen* - *WMaze* \* *sizekoef*) // 2  
  
  
# Клас для опису кожної клітинки лабіринту  
class Cell:  
 Walls = [0, 0, 0, 0] # Стіни клітинки (0 - немає стіни, 1 - є стіна; порядок: праворуч, вгору, ліворуч, вниз)  
 posx = 0 # Координати клітинки на екрані по осі x  
 posy = 0 # Координати клітинки на екрані по осі y  
 Cheese = 0 # Позначка для монет/сиру  
  
  
# Клас для опису ігрових об'єктів (Pacman, привиди)  
class Object:  
 start = -1 # Час початку руху  
 posx = 0 # Позиція на екрані по осі x  
 posy = 0 # Позиція на екрані по осі y  
 x = 0 # Позиція на сітці по осі x  
 y = 0 # Позиція на сітці по осі y  
 xprev = 0 # Попередня позиція по осі x  
 yprev = 0 # Попередня позиція по осі y  
 typ = "Pacman" # Тип об'єкта (Pacman або Ghost)  
 direct = 0 # Напрямок руху (0 - вправо, 1 - вгору, 2 - вліво, 3 - вниз)  
 color = (0, 0, 255) # Колір об'єкта (за замовчуванням синій)  
 number = -1 # Номер об'єкта (важливо для привидів)  
  
  
# Клас для відстеження зіткнень між об'єктами  
class Collision:  
 happened = False # Ознака, чи відбулося зіткнення  
  
  
# Функція для відображення графіки гри  
def graph():  
 *screen*.fill((0, 0, 0)) # Очищення екрану (чорний фон)  
  
 # Малювання стін для кожної клітинки лабіринту  
 for *row* in *Cells*:  
 for *c* in *row*:  
 # Якщо є стіна праворуч, малюємо її  
 if *c*.Walls[0] == 1:  
 pygame.draw.line(*screen*, (0, 255, 0), (*left* + *c*.posx + *sizekoef*, *top* + *c*.posy),  
 (*left* + *c*.posx + *sizekoef*, *top* + *c*.posy + *sizekoef*))  
 # Якщо є стіна вгору, малюємо її  
 if *c*.Walls[1] == 1:  
 pygame.draw.line(*screen*, (0, 255, 0), (*left* + *c*.posx, *top* + *c*.posy),  
 (*left* + *c*.posx + *sizekoef*, *top* + *c*.posy))  
 # Якщо є стіна ліворуч, малюємо її  
 if *c*.Walls[2] == 1:  
 pygame.draw.line(*screen*, (0, 255, 0), (*left* + *c*.posx, *top* + *c*.posy),  
 (*left* + *c*.posx, *top* + *c*.posy + *sizekoef*))  
 # Якщо є стіна вниз, малюємо її  
 if *c*.Walls[3] == 1:  
 pygame.draw.line(*screen*, (0, 255, 0), (*left* + *c*.posx, *top* + *c*.posy + *sizekoef*),  
 (*left* + *c*.posx + *sizekoef*, *top* + *c*.posy + *sizekoef*))  
  
 # Якщо потрібно показувати шлях, малюємо його  
 if *showpath*:  
 *m* = 1  
 for *i* in range(*WMaze*):  
 for *j* in range(*HMaze*):  
 *Entered*[*j*][*i*][0] = *Entered*[*j*][*i*][0] + 1  
 *m* = max(*m*, *Entered*[*j*][*i*][0])  
 # Малюємо всі відвідані клітинки різними відтінками сірого  
 for *i* in range(*WMaze*):  
 for *j* in range(*HMaze*):  
 *color* = (255 \* *Entered*[*j*][*i*][0] // *m*, 255 \* *Entered*[*j*][*i*][0] // *m*, 255 \* *Entered*[*j*][*i*][0] // *m*)  
 pygame.draw.circle(*screen*, *color*,  
 (*left* + *sizekoef* \* *i* + *sizekoef* // 2, *top* + *sizekoef* \* *j* + *sizekoef* // 2),  
 *sizekoef* // 3)  
 # Малюємо червоний шлях  
 for *c* in *path*:  
 pygame.draw.circle(*screen*, (255, 0, 0),  
 (*left* + *sizekoef* \* *c*[0] + *sizekoef* // 2, *top* + *sizekoef* \* *c*[1] + *sizekoef* // 2),  
 *sizekoef* // 3)  
  
 # Малюємо всі об'єкти гри (Pacman та привиди)  
 for *obj* in *objects*:  
 pygame.draw.circle(*screen*, *obj*.color, (  
 *left* + int(*sizekoef* \* *obj*.posx) + *sizekoef* // 2, *top* + int(*sizekoef* \* *obj*.posy) + *sizekoef* // 2),  
 *sizekoef* // 2)  
 # Перевіряємо зіткнення привидів з Pacman  
 if *obj*.typ == "Ghost" and abs(*obj*.posx - *player*.posx) + abs(*obj*.posy - *player*.posy) < 0.2:  
 *collision*.happened = True  
  
 # Малюємо монетку (coin)  
 pygame.draw.circle(*screen*, (255, 255, 0),  
 (*left* + *sizekoef* \* *coin*[0] + *sizekoef* // 2, *top* + *sizekoef* \* *coin*[1] + *sizekoef* // 2),  
 *sizekoef* // 4)  
  
  
# Функція для перевірки з'єднаності клітинок лабіринту  
def CheckConnection():  
 *used* = []  
 for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *row*.append(False)  
 *used*.append(*row*)  
  
 # Використовуємо пошук у ширину (BFS) для перевірки з'єднання  
 *q* = []  
 *q*.append((0, 0))  
 *used*[0][0] = True  
  
 while (len(*q*) > 0):  
 *pos* = *q*[0]  
 *x* = *pos*[0]  
 *y* = *pos*[1]  
 # Рухаємося вгору, вниз, вліво, вправо, якщо немає стін  
 if *Cells*[*x*][*y*].Walls[0] == 0 and not *used*[*x*][*y* + 1]:  
 *q*.append((*x*, *y* + 1))  
 *used*[*x*][*y* + 1] = True  
 if *Cells*[*x*][*y*].Walls[1] == 0 and not *used*[*x* - 1][*y*]:  
 *q*.append((*x* - 1, *y*))  
 *used*[*x* - 1][*y*] = True  
 if *Cells*[*x*][*y*].Walls[2] == 0 and not *used*[*x*][*y* - 1]:  
 *q*.append((*x*, *y* - 1))  
 *used*[*x*][*y* - 1] = True  
 if *Cells*[*x*][*y*].Walls[3] == 0 and not *used*[*x* + 1][*y*]:  
 *q*.append((*x* + 1, *y*))  
 *used*[*x* + 1][*y*] = True  
 *q*.pop(0)  
  
 # Перевіряємо, чи всі клітинки з'єднані  
 *ans* = True  
 for *i* in *used*:  
 for *j* in *i*:  
 *ans* = *ans* and *j* return *ans*# Функція генерації лабіринту за допомогою алгоритму DFS  
def dfs(*i*, *j*, *direc*):  
 *used*[*j*][*i*] = True  
 *possibles* = [0, 1, 2, 3] # Напрямки руху: вправо, вгору, вліво, вниз  
 random.shuffle(*possibles*) # Випадковий порядок  
  
 for *decision* in *possibles*:  
 # Рух вліво  
 if *decision* == 2:  
 if *i* > 0 and not *used*[*j*][*i* - 1]:  
 dfs(*i* - 1, *j*, 0)  
 *Cells*[*j*][*i*].Walls[*decision*] = 0  
 else:  
 *Cells*[*j*][*i*].Walls[2] = 1  
 # Рух вправо  
 if *decision* == 0:  
 if *i* < *WMaze* - 1 and not *used*[*j*][*i* + 1]:  
 dfs(*i* + 1, *j*, 2)  
 *Cells*[*j*][*i*].Walls[*decision*] = 0  
 else:  
 *Cells*[*j*][*i*].Walls[0] = 1  
 # Рух вгору  
 if *decision* == 1:  
 if *j* > 0 and not *used*[*j* - 1][*i*]:  
 dfs(*i*, *j* - 1, 3)  
 *Cells*[*j*][*i*].Walls[*decision*] = 0  
 else:  
 *Cells*[*j*][*i*].Walls[1] = 1  
 # Рух вниз  
 if *decision* == 3:  
 if *j* < *HMaze* - 1 and not *used*[*j* + 1][*i*]:  
 dfs(*i*, *j* + 1, 1)  
 *Cells*[*j*][*i*].Walls[*decision*] = 0  
 else:  
 *Cells*[*j*][*i*].Walls[3] = 1  
  
 if *direc* >= 0:  
 *Cells*[*j*][*i*].Walls[*direc*] = 0  
  
  
# Імовірність для визначення дій привидів  
def prob():  
 return math.exp(-*Difficulty* / 3)  
  
  
# Функція оцінки шляху для алгоритму A\*  
def prior(*x*, *y*, x1, y1, *x2*, *y2*):  
 if *search* == "G": # Жадібний пошук  
 return abs(*x* - *x2*) + abs(*y* - *y2*)  
 if *search* == "A\*": # A\*  
 return abs(*x* - *x2*) + abs(*y* - *y2*) + (*Entered*[*y*][*x*][0] \* 2) // 4  
  
  
# Пошук шляху між точками (x1, y1) і (x2, y2) з використанням A\* або жадібного пошуку  
def findPath(*x1*, *y1*, *x2*, *y2*, *Entered*):  
 for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *row*.append([-1, -1, -1]) # Ініціалізація масиву Entered  
 *Entered*.append(*row*)  
  
 *q* = []  
 heapq.heappush(*q*, (0, *x1*, *y1*, -1, -1)) # Додаємо початкову позицію  
  
 while len(*q*) > 0:  
 *pos* = heapq.heappop(*q*)  
 *x* = *pos*[1]  
 *y* = *pos*[2]  
 *xprev* = *pos*[3]  
 *yprev* = *pos*[4]  
 if *xprev* != -1:  
 *Entered*[*y*][*x*] = [*Entered*[*y*][*x*][0], *xprev*, *yprev*]  
 else:  
 *Entered*[*y*][*x*] = [0, -1, -1]  
  
 # Якщо досягли мети  
 if *x* == *x2* and *y* == *y2*:  
 break  
  
 # Перевіряємо напрямки для руху  
 if *x* > 0 and *Cells*[*y*][*x*].Walls[2] == 0 and *Entered*[*y*][*x* - 1][0] == -1:  
 *Entered*[*y*][*x* - 1][0] = *Entered*[*y*][*x*][0] + 1  
 heapq.heappush(*q*, (prior(*x* - 1, *y*, *x1*, *y1*, *x2*, *y2*), *x* - 1, *y*, *x*, *y*))  
 if *x* < *WMaze* - 1 and *Cells*[*y*][*x*].Walls[0] == 0 and *Entered*[*y*][*x* + 1][0] == -1:  
 *Entered*[*y*][*x* + 1][0] = *Entered*[*y*][*x*][0] + 1  
 heapq.heappush(*q*, (prior(*x* + 1, *y*, *x1*, *y1*, *x2*, *y2*), *x* + 1, *y*, *x*, *y*))  
 if *y* > 0 and *Cells*[*y*][*x*].Walls[1] == 0 and *Entered*[*y* - 1][*x*][0] == -1:  
 *Entered*[*y* - 1][*x*][0] = *Entered*[*y*][*x*][0] + 1  
 heapq.heappush(*q*, (prior(*x*, *y* - 1, *x1*, *y1*, *x2*, *y2*), *x*, *y* - 1, *x*, *y*))  
 if *y* < *HMaze* - 1 and *Cells*[*y*][*x*].Walls[3] == 0 and *Entered*[*y* + 1][*x*][0] == -1:  
 *Entered*[*y* + 1][*x*][0] = *Entered*[*y*][*x*][0] + 1  
 heapq.heappush(*q*, (prior(*x*, *y* + 1, *x1*, *y1*, *x2*, *y2*), *x*, *y* + 1, *x*, *y*))  
  
 # Відновлення шляху  
 *xans* = *x2  
 yans* = *y2  
 ans* = []  
 while *xans* != *x1* or *yans* != *y1*:  
 *ans*.insert(0, (*xans*, *yans*))  
 (*xans*, *yans*) = (*Entered*[*yans*][*xans*][1], *Entered*[*yans*][*xans*][2])  
  
 return *ans  
  
  
score* = 0  
*collision* = Collision()  
*collision*.happened = False  
pygame.init()  
*clock* = pygame.time.Clock()  
  
*screen* = pygame.display.set\_mode((*WScreen*, *HScreen*))  
pygame.display.set\_caption("PacMan")  
*screen*.fill((0, 0, 10))  
pygame.display.flip()  
  
# Створення лабіринту  
*Cells* = []  
for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *c* = Cell()  
 *c*.Walls = [0, 0, 0, 0]  
 *c*.posx = *j* \* *sizekoef  
 c*.posy = *i* \* *sizekoef  
 row*.append(*c*)  
 *Cells*.append(*row*)  
for *i* in range(*HMaze*):  
 *Cells*[*i*][0].Walls[2] = 1  
 *Cells*[*i*][*WMaze* - 1].Walls[0] = 1  
for *j* in range(*WMaze*):  
 *Cells*[0][*j*].Walls[1] = 1  
 *Cells*[*HMaze* - 1][*j*].Walls[3] = 1  
  
# Генерація лабіринту методом Wall-Set (WS) або DFS  
if *generation* == "WS":  
 *possiblewalls* = []  
 for *i* in range(*WMaze*):  
 for *j* in range(*HMaze*):  
 for *k* in range(2):  
 if random.uniform(0, 1) <= *walldensity*:  
 *possiblewalls*.append((*i*, *j*, *k*))  
 random.shuffle(*possiblewalls*)  
 for *w* in *possiblewalls*:  
 *i* = *w*[0]  
 *j* = *w*[1]  
 *t* = *w*[2]  
 if *t* == 0 and *i* < *WMaze* - 1:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j*][*i* + 1].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[0] = 1  
 *Cells*[*j*][*i* + 1].Walls[2] = 1  
 if not (CheckConnection()):  
 *Cells*[*j*][*i*].Walls[0] = 0  
 *Cells*[*j*][*i* + 1].Walls[2] = 0  
 if *t* == 1 and *j* > 0:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j* - 1][*i*].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[1] = 1  
 *Cells*[*j* - 1][*i*].Walls[3] = 1  
 if not (CheckConnection()):  
 *Cells*[*j*][*i*].Walls[1] = 0  
 *Cells*[*j* - 1][*i*].Walls[3] = 0  
 if *t* == 2 and *i* > 0:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j*][*i* - 1].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[2] = 1  
 *Cells*[*j*][*i* - 1].Walls[0] = 1  
 if not (CheckConnection()):  
 *Cells*[*j*][*i*].Walls[2] = 0  
 *Cells*[*j*][*i* - 1].Walls[0] = 0  
 if *t* == 3 and *j* < *HMaze* - 1:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j* + 1][*i*].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[3] = 1  
 *Cells*[*j* + 1][*i*].Walls[1] = 1  
 if not (CheckConnection()):  
 *Cells*[*j*][*i*].Walls[3] = 0  
 *Cells*[*j* + 1][*i*].Walls[1] = 0  
if *generation* == "DFS":  
 *used* = []  
 for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *row*.append(False)  
 *used*.append(*row*)  
 dfs(0, 0, -1)  
*Entered* = []  
*pacman* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
*coin* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
while abs(*pacman*[0] - *coin*[0]) + abs(*pacman*[1] - *coin*[1]) <= (*WMaze* // 4):  
 *coin* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
*path* = findPath(*pacman*[0], *pacman*[1], *coin*[0], *coin*[1], *Entered*)  
*running* = True  
*arrow* = ""  
*objects* = []  
*player* = Object()  
*player*.x = *pacman*[0]  
*player*.y = *pacman*[1]  
*player*.xprev = *player*.posx  
*player*.yprev = *player*.posy  
*player*.typ = "Pacman"  
*player*.color = (255, 255, 0)  
*objects*.append(*player*)  
*gn* = 0  
*first* = None  
for *i* in range(*ghostnumber*):  
 *gh* = Object()  
 *gh*.x = random.randint(0, *WMaze* - 1)  
 *gh*.y = random.randint(0, *HMaze* - 1)  
 while abs(*pacman*[0] - *gh*.x) + abs(*pacman*[1] - *gh*.y) <= (*WMaze* // 4):  
 *gh*.x = random.randint(0, *WMaze* - 1)  
 *gh*.y = random.randint(0, *HMaze* - 1)  
 *gh*.xprev = *gh*.x  
 *gh*.yprev = *gh*.y  
 *gh*.typ = "Ghost"  
 *gh*.color = (random.randint(0, 255), random.randint(0, 255), 255)  
 *gh*.number = *gn* if *gn* == 0:  
 *first* = *gh  
 gn* += 1  
 *objects*.append(*gh*)  
  
  
def handleCellsWS():  
 *Cells*.clear()  
  
  
def handleCellsDFS():  
 Cells = []  
  
  
# Global variables for levels and difficulty adjustments  
*current\_level* = 1  
*ghost\_increment* = 2 # Increase ghost number by 2 each level  
*difficulty\_increment* = 20 # Increase difficulty by 20 each level  
  
  
# Function to reset the game for the next level  
def next\_level():  
 global *ghostnumber*, *Difficulty*, *player*, *objects*, *pacman*, *coin*, *path*, *current\_level  
  
 current\_level* += 1  
 *ghostnumber* += *ghost\_increment* # Increase ghost number  
 *Difficulty* += *difficulty\_increment* # Increase difficulty  
  
 # Regenerate maze  
 # Instead of Cells.clear() or Cells = [], let's recreate the grid explicitly  
  
 if *generation* == "WS":  
 handleCellsWS()  
  
 if *generation* == "DFS":  
 handleCellsDFS()  
  
 for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *c* = Cell()  
 *c*.Walls = [0, 0, 0, 0]  
 *c*.posx = *j* \* *sizekoef  
 c*.posy = *i* \* *sizekoef  
 row*.append(*c*)  
 *Cells*.append(*row*)  
 for *i* in range(*HMaze*):  
 *Cells*[*i*][0].Walls[2] = 1  
 *Cells*[*i*][*WMaze* - 1].Walls[0] = 1  
 for *j* in range(*WMaze*):  
 *Cells*[0][*j*].Walls[1] = 1  
 *Cells*[*HMaze* - 1][*j*].Walls[3] = 1  
  
 if *generation* == "WS":  
 *possiblewalls* = []  
 for *i* in range(*WMaze*):  
 for *j* in range(*HMaze*):  
 for *k* in range(2):  
 if random.uniform(0, 1) <= *walldensity*:  
 *possiblewalls*.append((*i*, *j*, *k*))  
 random.shuffle(*possiblewalls*)  
 for *w* in *possiblewalls*:  
 *i* = *w*[0]  
 *j* = *w*[1]  
 *t* = *w*[2]  
 if *t* == 0 and *i* < *WMaze* - 1:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j*][*i* + 1].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[0] = 1  
 *Cells*[*j*][*i* + 1].Walls[2] = 1  
 if not CheckConnection():  
 *Cells*[*j*][*i*].Walls[0] = 0  
 *Cells*[*j*][*i* + 1].Walls[2] = 0  
 if *t* == 1 and *j* > 0:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j* - 1][*i*].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[1] = 1  
 *Cells*[*j* - 1][*i*].Walls[3] = 1  
 if not CheckConnection():  
 *Cells*[*j*][*i*].Walls[1] = 0  
 *Cells*[*j* - 1][*i*].Walls[3] = 0  
 if *t* == 2 and *i* > 0:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j*][*i* - 1].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[2] = 1  
 *Cells*[*j*][*i* - 1].Walls[0] = 1  
 if not CheckConnection():  
 *Cells*[*j*][*i*].Walls[2] = 0  
 *Cells*[*j*][*i* - 1].Walls[0] = 0  
 if *t* == 3 and *j* < *HMaze* - 1:  
 if sum(*Cells*[*j*][*i*].Walls) < 2 and sum(*Cells*[*j* + 1][*i*].Walls) < 2:  
 *Cells*[*j*][*i*].Walls[3] = 1  
 *Cells*[*j* + 1][*i*].Walls[1] = 1  
 if not CheckConnection():  
 *Cells*[*j*][*i*].Walls[3] = 0  
 *Cells*[*j* + 1][*i*].Walls[1] = 0  
 if *generation* == "DFS":  
 *used* = []  
 for *i* in range(*HMaze*):  
 *row* = []  
 for *j* in range(*WMaze*):  
 *row*.append(False)  
 *used*.append(*row*)  
 dfs(0, 0, -1)  
  
 # Clear objects and set up new game state  
 *objects* = []  
  
 # Reset Pacman's position  
 *pacman* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
 *player* = Object()  
 *player*.x = *pacman*[0]  
 *player*.y = *pacman*[1]  
 *player*.xprev = *player*.posx  
 *player*.yprev = *player*.posy  
 *player*.typ = "Pacman"  
 *player*.color = (255, 255, 0)  
 *objects*.append(*player*)  
  
 # Reset coin position  
 *coin* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
 while abs(*pacman*[0] - *coin*[0]) + abs(*pacman*[1] - *coin*[1]) <= (*WMaze* // 4):  
 *coin* = (random.randint(0, *WMaze* - 1), random.randint(0, *HMaze* - 1))  
  
 # Reset ghosts with increased number  
 for *i* in range(*ghostnumber*):  
 *gh* = Object()  
 *gh*.x = random.randint(0, *WMaze* - 1)  
 *gh*.y = random.randint(0, *HMaze* - 1)  
 while abs(*pacman*[0] - *gh*.x) + abs(*pacman*[1] - *gh*.y) <= (*WMaze* // 4):  
 *gh*.x = random.randint(0, *WMaze* - 1)  
 *gh*.y = random.randint(0, *HMaze* - 1)  
 *gh*.xprev = *gh*.x  
 *gh*.yprev = *gh*.y  
 *gh*.typ = "Ghost"  
 *gh*.color = (random.randint(0, 255), random.randint(0, 255), 255)  
 *gh*.number = *i  
 objects*.append(*gh*)  
  
 # Reset the path for the new level  
 *Entered* = []  
 *path* = findPath(*pacman*[0], *pacman*[1], *coin*[0], *coin*[1], *Entered*)  
  
  
# Основний цикл гри  
while *running*:  
 for *event* in pygame.event.get():  
 if *event*.type == pygame.QUIT:  
 *running* = False  
 elif *event*.type == pygame.KEYDOWN:  
 if *event*.key == pygame.K\_DOWN:  
 *arrow* = "Down"  
 if *event*.key == pygame.K\_UP:  
 *arrow* = "Up"  
 if *event*.key == pygame.K\_LEFT:  
 *arrow* = "Left"  
 if *event*.key == pygame.K\_RIGHT:  
 *arrow* = "Right"  
 # Обробка руху привидів  
 for *obj* in *objects*:  
 if *obj*.typ == "Ghost":  
 if *obj*.start == -1 or pygame.time.get\_ticks() - *obj*.start >= 600:  
 if random.uniform(0, 1) < prob():  
 *obj*.start = pygame.time.get\_ticks()  
 *obj*.xprev = *obj*.x  
 *obj*.yprev = *obj*.y  
 *d* = *obj*.direct  
 if *Cells*[*obj*.y][*obj*.x].Walls[*d*] == 1:  
 *r* = random.randint(0, 1)  
 if *r* == 0:  
 *d* = *d* + 1  
 *d* = *d* % 4  
 if *Cells*[*obj*.y][*obj*.x].Walls[*d*] == 1:  
 *d* = *d* + 2  
 *d* = *d* % 4  
 if *Cells*[*obj*.y][*obj*.x].Walls[*d*] == 1:  
 *d* = *d* + 3  
 *d* = *d* % 4  
 if *r* == 1:  
 *d* = *d* + 3  
 *d* = *d* % 4  
 if *Cells*[*obj*.y][*obj*.x].Walls[*d*] == 1:  
 *d* = *d* + 2  
 *d* = *d* % 4  
 if *Cells*[*obj*.y][*obj*.x].Walls[*d*] == 1:  
 *d* = *d* + 1  
 *d* = *d* % 4  
 *obj*.direct = *d* if *d* == 0:  
 *obj*.x = *obj*.x + 1  
 elif *d* == 1:  
 *obj*.y = *obj*.y - 1  
 elif *d* == 2:  
 *obj*.x = *obj*.x - 1  
 elif *d* == 3:  
 *obj*.y = *obj*.y + 1  
 else:  
 *obj*.start = pygame.time.get\_ticks()  
 *obj*.xprev = *obj*.x  
 *obj*.yprev = *obj*.y  
 *temp* = []  
 *temppath* = findPath(*obj*.x, *obj*.y, *player*.x, *player*.y, *temp*)  
 *temp* = []  
 *playerpath* = findPath(*player*.x, *player*.y, *coin*[0], *coin*[1], *temp*)  
 if len(*temppath*) > 0:  
 if *obj*.number == 0:  
 *obj*.x = *temppath*[0][0]  
 *obj*.y = *temppath*[0][1]  
 elif *obj*.number == 1:  
 if abs(*obj*.x - *player*.x) + abs(*obj*.y - *player*.y) > 3:  
 *targetx* = *player*.x - (*player*.y - *first*.y)  
 *targety* = *player*.y + (*player*.x - *first*.x)  
 else:  
 *targetx* = *player*.x  
 *targety* = *player*.y  
 if *targetx* < 0:  
 *targetx* = 0  
 if *targety* < 0:  
 *targety* = 0  
 if *targety* >= *HMaze*:  
 *targety* = *HMaze* - 1  
 if *targetx* >= *WMaze*:  
 *targetx* = *WMaze* - 1  
 *temp* = []  
 *temppath* = findPath(*obj*.x, *obj*.y, *targetx*, *targety*, *temp*)  
 if len(*temppath*) > 0:  
 *obj*.x = *temppath*[0][0]  
 *obj*.y = *temppath*[0][1]  
 elif *obj*.number == 2:  
 if abs(*obj*.x - *player*.x) + abs(*obj*.y - *player*.y) > 3:  
 *targetx* = *player*.x + (*player*.y - *first*.y)  
 *targety* = *player*.y - (*player*.x - *first*.x)  
 else:  
 *targetx* = *player*.x  
 *targety* = *player*.y  
 if *targetx* < 0:  
 *targetx* = 0  
 if *targety* < 0:  
 *targety* = 0  
 if *targety* >= *HMaze*:  
 *targety* = *HMaze* - 1  
 if *targetx* >= *WMaze*:  
 *targetx* = *WMaze* - 1  
 *temp* = []  
 *temppath* = findPath(*obj*.x, *obj*.y, *targetx*, *targety*, *temp*)  
 if (len(*temppath*) > 0):  
 *obj*.x = *temppath*[0][0]  
 *obj*.y = *temppath*[0][1]  
 elif (len(*temppath*) <= 4):  
 *obj*.x = *temppath*[0][0]  
 *obj*.y = *temppath*[0][1]  
 else:  
 *point* = len(*playerpath*) // (*ghostnumber* - 3)  
 *point* \*= (*obj*.number - 2)  
 *point* = len(*playerpath*) - *point  
 targetx* = 0  
 *targety* = 0  
 if (*point* < len(*playerpath*) and *point* >= 0):  
 *targetx* = *playerpath*[*point*][0]  
 *targety* = *playerpath*[*point*][1]  
 else:  
 *targetx* = *player*.x  
 *targety* = *player*.y  
 *targetx* += random.randint(0, 2) - 1  
 *targety* += random.randint(0, 2) - 1  
 if (*targetx* < 0):  
 *targetx* = 0  
 if (*targety* < 0):  
 *targety* = 0  
 if (*targety* >= *HMaze*):  
 *targety* = *HMaze* - 1  
 if (*targetx* >= *WMaze*):  
 *targetx* = *WMaze* - 1  
 *temp* = []  
 *temppath* = findPath(*obj*.x, *obj*.y, *targetx*, *targety*, *temp*)  
 if (len(*temppath*) > 0):  
 *obj*.x = *temppath*[0][0]  
 *obj*.y = *temppath*[0][1]  
  
 *obj*.posx = (((max(0, 600 - pygame.time.get\_ticks() + *obj*.start)) / 600) \* *obj*.xprev + (  
 ((min(600, pygame.time.get\_ticks() - *obj*.start)) / 600) \* *obj*.x))  
 *obj*.posy = (((max(0, 600 - pygame.time.get\_ticks() + *obj*.start)) / 600) \* *obj*.yprev + (  
 ((min(600, pygame.time.get\_ticks() - *obj*.start)) / 600) \* *obj*.y))  
  
 # Обробка руху пакмена  
 if *obj*.typ == "Pacman":  
 if *obj*.start == -1 or pygame.time.get\_ticks() - *obj*.start >= 400:  
 if *obj*.x == *coin*[0] and *obj*.y == *coin*[1]:  
 *score* += 1  
 print("Score: " + str(*score*))  
 next\_level() # Move to the next level when coin is collected  
 *obj*.start = pygame.time.get\_ticks()  
 *obj*.xprev = *obj*.x  
 *obj*.yprev = *obj*.y  
 if *arrow* == "Right" and *Cells*[*obj*.y][*obj*.x].Walls[0] == 0:  
 *obj*.x = *obj*.x + 1  
 *obj*.direct = 0  
 elif *arrow* == "Up" and *Cells*[*obj*.y][*obj*.x].Walls[1] == 0:  
 *obj*.y = *obj*.y - 1  
 *obj*.direct = 1  
 elif *arrow* == "Left" and *Cells*[*obj*.y][*obj*.x].Walls[2] == 0:  
 *obj*.x = *obj*.x - 1  
 *obj*.direct = 2  
 elif *arrow* == "Down" and *Cells*[*obj*.y][*obj*.x].Walls[3] == 0:  
 *obj*.y = *obj*.y + 1  
 *obj*.direct = 3  
 elif *arrow* != "" and *Cells*[*obj*.y][*obj*.x].Walls[*obj*.direct] == 0:  
 if *obj*.direct == 0:  
 *obj*.x = *obj*.x + 1  
 *arrow* = "Right"  
 if *obj*.direct == 1:  
 *obj*.y = *obj*.y - 1  
 *arrow* = "Up"  
 if *obj*.direct == 2:  
 *obj*.x = *obj*.x - 1  
 *arrow* = "Left"  
 if *obj*.direct == 3:  
 *obj*.y = *obj*.y + 1  
 *arrow* = "Down"  
 else:  
 if ((*arrow* == "Left" and *obj*.direct == 0) or (*arrow* == "Up" and *obj*.direct == 3) or (  
 *arrow* == "Down" and *obj*.direct == 1)):  
 *obj*.xprev, *obj*.x = *obj*.x, *obj*.xprev  
 *obj*.yprev, *obj*.y = *obj*.y, *obj*.yprev  
 *obj*.start = pygame.time.get\_ticks() - (400 - pygame.time.get\_ticks() + *obj*.start)  
 *obj*.direct += 2  
 *obj*.direct %= 4  
  
 *Entered* = []  
 *path* = findPath(*obj*.x, *obj*.y, *coin*[0], *coin*[1], *Entered*)  
 *obj*.posx = (((max(0, 400 - pygame.time.get\_ticks() + *obj*.start)) / 400) \* *obj*.xprev + (  
 ((min(400, pygame.time.get\_ticks() - *obj*.start)) / 400) \* *obj*.x))  
 *obj*.posy = (((max(0, 400 - pygame.time.get\_ticks() + *obj*.start)) / 400) \* *obj*.yprev + (  
 ((min(400, pygame.time.get\_ticks() - *obj*.start)) / 400) \* *obj*.y))  
  
 # колізії з привидами перевірка  
 for *obj* in *objects*:  
 if *obj*.typ == "Ghost" and *obj*.x == *player*.x and *obj*.y == *player*.y:  
 print("GAME OVER")  
 *running* = False  
 break  
  
 graph()  
 pygame.display.flip()