Cat & Mouse

Q: 3 мачки фаќаат 3 глувци за 3 мин. За колку време ќе се фатат 100 глувци?

Ова прашњае на прв поглед изгледа лесно и користјчи алгебра моземе да стигнеме до одговор од 100 мин. Но прашањето е дали преку симулација можеме да го докжеме тоа?

Некој услови за ова симулација ни се:

- NxP праваголно поле
- Секое животно фаќа една ќелија
- Иницијално случајно распоредени
- Едно движење трае една секунда
- Дефиниција на фаќање – мачка и глушец во иста ќелија

Спорд овје ограницуванја ние треба да создадеме симулација каде 3 мачки фаќаат 3 глувци за 3 мин. Потребно е да ги подесиме цледните параметри за да го добиеме посакуваниот резултат:

- N и P
- Брзина на глувци
- Брзина на мачки
- Стратегии за фаќање глувци
- Стратегии за бегање од мачки

Најтешкиот дел за програмирање и подоцна менување се стратегиите, така да:

- Глувците немат стратегија и се двизат случајно
- Додека мачките секогаш го бркат најблискиот глушец (Ова е вака затаоа како студенти од прва година не баш добро се разбираме во алгоритми)

Што значи ни престанува:

- Голеемина
- Брзина на глувци
- Брзиан на Мачки

За глувците да имат шанси потребно е мацките да бидат поспори од нив. Во овој случај Брзина на Глушец = 3 * Брзиан на Мачка (Од току зависи и големинта што после ќе ни биде потрбна)

Потоа прку изпробување моземе да ја најеџме потребната големина. Во овој случај 56

In [3]:
```
from Simulation import run_simulation

NUM_CATS=3
NUM_MICE=3

run_simulation(N=56, P=56, num_cats=NUM_CATS, num_mice=NUM_MICE, num_obstacl
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit
C:\Users\royal\AppData\Local\Programs\Python\Python39\lib\site-packages\IPyt
hon\core\interactiveshell.py:3558: UserWarning: To exit: use 'exit', 'quit',
or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

Поради случајната расподелба на мачки и глувци неможеме да добиеме конкретни резулати поради големата варијација. Затоа можеме да извршиме поголем број на симулација и да го земеме просечното време

In [1]:
```
from Simulation import run_simulation

NUM_CATS=3
NUM_MICE=3

number_of_runs=10
sum_of_turns=0
for x in range(number_of_runs):
    turns=run_simulation(N=56, P=56, num_cats=NUM_CATS, num_mice=NUM_MICE, r
    sum_of_turns += turns
    print(f"run {x} time in seconds {turns}")
print(f"Avrage time {sum_of_turns / number_of_runs}")
```

```
  Cell In[1], line 3
    NUM_CATS=3p
             ^
SyntaxError: invalid syntax
```

Со изработка на 10000 симулација и наочање на нивниот просек добиваме резултат од 184 секунди ли 3 мин и 4 сек. Резулатаи од 3 Мачки и 3 Глувци

Откако знаме дека нашата калибрација е во корисна состојба може да ранаме симулација со 100 глувци.

In [1]:
```
from Simulation import run_simulation

NUM_CATS=3
NUM_MICE=100

run_simulation(N=56, P=56, num_cats=NUM_CATS, num_mice=NUM_MICE, num_obstacl
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.9.0)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Out[1]: 1524

Исто како кај случајот со 3 глувци треба да ја извршиме симулацијата голема број на пати. Во овој слуцај добиваме дека резултато е приближно 1389 секунди или 23 мин и 9 сек. Резулатаи од 3 Мачки и 100 Глувци

In [ ]:
```
from Simulation import run_simulation

NUM_CATS=3
NUM_MICE=100

number_of_runs=10
sum_of_turns=0
for x in range(number_of_runs):
    turns=run_simulation(N=56, P=56, num_cats=NUM_CATS, num_mice=NUM_MICE, r
    sum_of_turns += turns
    print(f"run {x} time in seconds {turns}")
print(f"Avrage time {sum_of_turns / number_of_runs}")
```

Симулација

Наредното е главниот код кој во суштина ги изработува симулацијите (Не е препорачано до го ранате)

In [ ]:
```
import pygame
import sys
import random
import time
from Cat import Cat
from Mouse import Mouse

# Самата симулација е дефинирана како функција
# Функцијата го прима бројот на мачки и глувци, големината, број на препреки
def run_simulation(N=80, P=80, num_cats=3, num_mice=3, num_obstacles=150, wa
    # Се дефинира големината на целија и големината на прозорецот
    CELL_SIZE = 10
    WIDTH, HEIGHT = P * CELL_SIZE, N * CELL_SIZE

    # Се дефинират разлицни бои за различни променливи
    WHITE = (255, 255, 255)
    GRAY = (200, 200, 200)
    CAT_COLOR = (255, 100, 100)
    MOUSE_COLOR = (100, 100, 255)
    TEXT_COLOR = (0, 0, 0)
    WIN_COLOR = (0, 200, 0)

    # Иницијализирај pygame
    # Користиме pygame за да ни го олесни симулацијата и визулацијата
    screen = pygame.display.set_mode((WIDTH, HEIGHT + 40))
    pygame.display.set_caption("Cat and Mouse")
```

```
clock = pygame.time.Clock()
font = pygame.font.SysFont(None, 30)

# Иницилизираме низи за Мачките, Глувците и Препреките
cats = []
mice = []
occupied = set()
obstacles = set()

# Ги поставуваме сите елементи на случајни места и ги додаваме во содвет
for _ in range(num_cats):
    while True:
        r, c = random.randint(0, N - 1), random.randint(0, P - 1)
        if (r, c) not in occupied:
            cats.append(Cat(r, c))
            occupied.add((r, c))
            break

for _ in range(num_mice):
    while True:
        r, c = random.randint(0, N - 1), random.randint(0, P - 1)
        if (r, c) not in occupied:
            mice.append(Mouse(r, c))
            occupied.add((r, c))
            break

while len(obstacles) < num_obstacles:
    r, c = random.randint(0, N - 1), random.randint(0, P - 1)
    if (r, c) not in occupied:
        obstacles.add((r, c))
        occupied.add((r, c))

# Бидејчи секое движење е една секунда ги броиме потребните циклуси
turns = 0

# Дефинираме функции за визуализација на елементите
def draw_grid():
    for row in range(N):
        for col in range(P):
            rect = pygame.Rect(col * CELL_SIZE, row * CELL_SIZE, CELL_SI
            pygame.draw.rect(screen, GRAY, rect, 1)

def draw_objects():
    for cat in cats:
        row, col = cat.get_position()
        rect = pygame.Rect(col * CELL_SIZE, row * CELL_SIZE, CELL_SIZE,
        pygame.draw.rect(screen, CAT_COLOR, rect)

    for mouse in mice:
        row, col = mouse.get_position()
        rect = pygame.Rect(col * CELL_SIZE, row * CELL_SIZE, CELL_SIZE,
        pygame.draw.rect(screen, MOUSE_COLOR, rect)

    for (row, col) in obstacles:
        rect = pygame.Rect(col * CELL_SIZE, row * CELL_SIZE, CELL_SIZE,
        pygame.draw.rect(screen, (50, 50, 50), rect)
```

```
    def draw_turn_count():
        text = font.render(f"Turn: {turns}", True, TEXT_COLOR)
        screen.blit(text, (10, HEIGHT + 10))

    # Ова е главниот луп на симулацијата
    while True:
        screen.fill(WHITE)
        draw_grid()
        draw_objects()
        draw_turn_count()

        pygame.display.flip()

        # Се чека крај на симулацијата за да се затвори прозорот
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

        # Цекање помеѓу циклуси
        time.sleep(wait_time/5000)

        # Мачикете се движат на секој 3 циклус кон најблискиут Глушеч
        #( Повеќе објацнување кај Мачките)
        if (turns % 3) == 0:
            new_cat_positions = set()
            for cat in cats:
                others = new_cat_positions | {c.get_position() for c in cats
                cat.move_toward_mouse(mice, N, P, obstacles, others)
                new_cat_positions.add(cat.get_position())

        # Проверка дали некој глушец дели позиција со Мачка, и него отстрану
        cat_positions = {cat.get_position() for cat in cats}
        mice = [m for m in mice if m.get_position() not in cat_positions]

        # Движење на глувците
        new_mouse_positions = set()
        for mouse in mice:
            others = new_mouse_positions | {m.get_position() for m in mice i
            mouse.move_random(N, P, obstacles, others)
            new_mouse_positions.add(mouse.get_position())

        cat_positions = {cat.get_position() for cat in cats}
        mice = [m for m in mice if m.get_position() not in cat_positions]

        # Инкрементација на циклуси
        turns += 1

        # Проверка дали сеуште има глувци, ако нема се врача бројот на циклу
        if not mice or turns>12000:
            pygame.display.flip()
            time.sleep(wait_time)
            pygame.quit()
            return turns
```

```
        clock.tick(10)
```

Глушец

Ќе почнемо со објаснување на кодот за Глушецот. Глувците се движзат случајно, ова е наједноставното решение за глувците

In [ ]:
```
import random
class Mouse:
    # Иницализација на Глушецот
    def __init__(self, row, col):
        self.row = row
        self.col = col

    def get_position(self):
        return self.row, self.col

    def move_random(self, max_rows, max_cols, obstacles=set(), occupied=set(
        # Глушецот пробува да најде случајна позиција на која може да се пом
        # Ако прват позиција не е слободна ќе продолжи да тражи додека ненај
        tried = set()
        while len(tried) < 4:
            direction = random.randint(0, 3)
            if direction in tried:
                continue
            tried.add(direction)

            new_r, new_c = self.row, self.col
            if direction == 0 and self.row > 0:
                new_r -= 1
            elif direction == 1 and self.row < max_rows - 1:
                new_r += 1
            elif direction == 2 and self.col > 0:
                new_c -= 1
            elif direction == 3 and self.col < max_cols - 1:
                new_c += 1

            # Кога ќе најде слободна позиција се поместува на неаА
            if (new_r, new_c) not in obstacles and (new_r, new_c) not in occ
                self.row, self.col = new_r, new_c
                break
```

Мачка

Кодот за мачките е малку покомплициран од тој од Глувците. Мачките секогаш го бркат најблискиот глушец, ова е стратгијата на мацките поради едноставноста на програмирање.

In [ ]:
```
from pathfinding import a_star, manhattan
```

```
class Cat:
    # Иницализација на Мачка
    def __init__(self, row, col):
        self.row = row
        self.col = col


    def get_position(self):
        return self.row, self.col

    def move_toward_mouse(self, mice, grid_rows, grid_cols, obstacles=set(),
        if not mice:
            return

        my_pos = self.get_position()

        # Мачката го наоча најблискит Глушец
        nearest = min(mice, key=lambda m: manhattan(my_pos, m.get_position()
        target_pos = nearest.get_position()

        # Го наоча патот до најблискиот Глушец
        path = a_star(my_pos, target_pos, grid_rows, grid_cols, obstacles |

        # Се движи кон најблискиот Глушец ако патот е слободен
        if path and path[0] not in occupied:
            self.row, self.col = path[0]
```

Пронаоѓање на пат

Штом Мачките секогаш го бркат најблискиот глушец ни треба неговата локација и најкраткиот пат до него.

In [ ]:
```
import heapq

# Ја пресметува дистанцата измечу две цели
def manhattan(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Користиме A* за да го најдеме најбрзиот пат до Глушецот
# ( Искрено го гледав од YouTube па ќе повторим нешто што слушнав)
def a_star(start, goal, grid_rows, grid_cols, obstacles=set()):
    open_set = []
    heapq.heappush(open_set, (0 + manhattan(start, goal), 0, start))
    came_from = {}
    cost_so_far = {start: 0}

    while open_set:
        _, cost, current = heapq.heappop(open_set)

        # Кога ќе ја најдеме челта го врачаме патот
        if current == goal:
            path = []
            while current != start:
                path.append(current)
```

```
                current = came_from[current]
            path.reverse()
            return path

        # Ги проверува сите цели до моментална целија
        for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
            neighbor = (current[0] + dr, current[1] + dc)
            if 0 <= neighbor[0] < grid_rows and 0 <= neighbor[1] < grid_cols
                new_cost = cost + 1
                # Пресметува која целија е најблиску до целта и ја се постав
                # Истовремено ја запишува целијата од каде е дојден
                if neighbor not in cost_so_far or new_cost < cost_so_far[nei
                    cost_so_far[neighbor] = new_cost
                    priority = new_cost + manhattan(neighbor, goal)
                    heapq.heappush(open_set, (priority, new_cost, neighbor))
                    came_from[neighbor] = current

    return []
```

Резултаи

Како краен чекор за симулацијата ги запишуваме резултатите во Ексел документ

In [ ]:
```
from Simulation import run_simulation
import openpyxl

NUM_MICE=3
NUM_CATS=3

# Креираме и отвараме нов Ексел фајл
wb = openpyxl.Workbook()
sheet = wb.active
sum=0

# Казуваме колку симулации ни се потребни
number_of_runs=10

sheet["A1"]="Cats:"
sheet["B1"]="Mouse:"
sheet["C1"]="Time(sec):"

# Ја извршуваме симулацијата одреден број на пати
for x in range(number_of_runs):
    turns = run_simulation(N=56, P=56, num_cats=NUM_CATS, num_mice=NUM_MICE,
    sum+=turns
    print(f"{x}: Game finished in {turns} turns!")
    # После секоја симулација ги запишуваме резултати
    sheet["A"+str((x+2))]=NUM_CATS
    sheet["B"+str((x+2))]=NUM_MICE
    sheet["C"+str((x+2))]=turns
print(f"Avrage time {sum/number_of_runs} turns!")
```

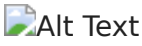
```
# Го зачувуваме Ексел фајлот
wb.save("primer.xlsx")
```

Останато

Линкови до користени ресурси

- PyGame
- Pathfinding
- OpenPyxl

Следното е граф од потребно време за 3 мачки до фата n глувци, каде секое време е просек од 100 симулации



This notebook was converted with convert.ploomber.io