Advanced Robotics Tutorial: Intelligent Game-Playing Bot

Overview

In this advanced robotics tutorial, we design a smart game-playing bot that navigates a grid environment with dynamic coin placement and static obstacles. The bot uses the A* path planning algorithm to move optimally towards the nearest coin while avoiding obstacles. This tutorial integrates robotics concepts such as state-based planning, autonomous pathfinding, and intelligent environment interaction.

Key Features

- Grid-based 2D game world
- A* algorithm for optimal pathfinding
- Dynamic coin respawning
- Robot collects coins and avoids static obstacles

Python Code

```
import pygame
import heapq
import random

# Constants
WIDTH, HEIGHT = 600, 600
ROWS, COLS = 20, 20
CELL_SIZE = WIDTH // COLS

WHITE, BLACK, GREEN, RED, YELLOW, GREY = (255,255,255), (0,0,0), (0,255,0), (255,0,0), (255,255,0), (200,200,200)

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Advanced Game-Playing Robot")
clock = pygame.time.Clock()

# Environment
robot = (0, 0)
coins = [(random.randint(0, ROWS-1), random.randint(0, COLS-1)) for _ in range(3)]
obstacles = {(10, i) for i in range(5, 15)}

def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def get_neighbors(node):
    r, c = node
    for dr, dc in [(-1,0), (1,0), (0,-1), (0,1)]:
```

Advanced Robotics Tutorial: Intelligent Game-Playing Bot

```
        nr, nc = r + dr, c + dc
        if 0 <= nr < ROWS and 0 <= nc < COLS:
            yield (nr, nc)

def a_star(start, goal):
    open_set = []
    heapq.heappush(open_set, (heuristic(start, goal), 0, start, []))
    visited = set()

    while open_set:
        _, cost, current, path = heapq.heappop(open_set)
        if current in visited:
            continue
        visited.add(current)

        if current == goal:
            return path + [current]

        for neighbor in get_neighbors(current):
            if neighbor in obstacles:
                continue
              heapq.heappush(open_set, (cost + 1 + heuristic(neighbor, goal), cost + 1, neighbor,
path + [current]))
    return []

def draw_grid():
    for x in range(0, WIDTH, CELL_SIZE):
        pygame.draw.line(screen, GREY, (x, 0), (x, HEIGHT))
    for y in range(0, HEIGHT, CELL_SIZE):
        pygame.draw.line(screen, GREY, (0, y), (WIDTH, y))

def draw_game():
    screen.fill(WHITE)
    draw_grid()

    for (r, c) in obstacles:
        pygame.draw.rect(screen, BLACK, (c*CELL_SIZE, r*CELL_SIZE, CELL_SIZE, CELL_SIZE))

    for (r, c) in coins:
        pygame.draw.rect(screen, YELLOW, (c*CELL_SIZE, r*CELL_SIZE, CELL_SIZE, CELL_SIZE))

          pygame.draw.rect(screen,  GREEN,  (robot[1]*CELL_SIZE,  robot[0]*CELL_SIZE,  CELL_SIZE,
CELL_SIZE))

def respawn_coin():
    while True:
        new_coin = (random.randint(0, ROWS-1), random.randint(0, COLS-1))
        if new_coin not in obstacles and new_coin != robot:
            coins.append(new_coin)
            break
```

Advanced Robotics Tutorial: Intelligent Game-Playing Bot

```
def main():
    global robot
    running = True
    path = []
    goal = None

    while running:
        if not coins:
            respawn_coin()

        if not path:
            goal = min(coins, key=lambda coin: heuristic(robot, coin))
            path = a_star(robot, goal)

        if path:
            robot = path.pop(0)

        if robot in coins:
            coins.remove(robot)

        draw_game()
        pygame.display.flip()
        clock.tick(5)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

    pygame.quit()

if __name__ == "__main__":
    main()
```