Design and Implementation of an AI-Driven Game Character

Victoria Reed @aicompetence.org

July 27, 2024

1 Introduction

This document outlines the design and implementation of an AI-driven game character. The character will use basic AI techniques to interact with the game environment and players, demonstrating decision-making capabilities and adaptive behavior.

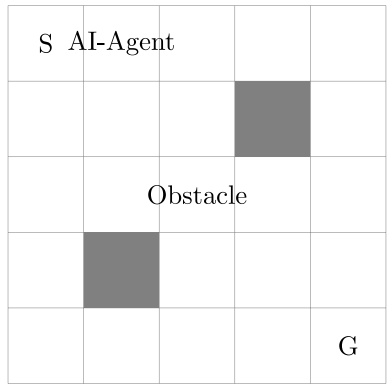
2 Character Design

The game character, named *AI-Agent*, is designed to navigate a simple grid-based environment, avoid obstacles, and interact with the player. The primary components of the AI-Agent are:

- Perception: The ability to perceive the environment.
- Decision Making: The ability to make decisions based on perceptions.
- Actions: The ability to execute actions within the environment.

3 Environment Setup

The environment is a grid-based world where the AI-Agent can move and interact. Obstacles are placed in the grid to create challenges for the AI-Agent.



4 AI Techniques

The AI-Agent employs the following techniques:

- **Pathfinding:** Using the A* algorithm to find the optimal path to a target.
- **Finite State Machines (FSM):** To manage the character's state transitions.
- **Behavior Trees:** To define complex behavior in a hierarchical manner.

5 Implementation

The AI-Agent is implemented in Python. Below is the code that demonstrates the key components.

5.1 Environment Setup

Listing 1: Environment Setup

```
import numpy as np

class Environment:
    def __init__(self, grid_size):
        self.grid_size = grid_size
        self.grid = np.zeros((grid_size, grid_size))

    def add_obstacle(self, position):
        self.grid[position] = 1

    def is_obstacle(self, position):
        return self.grid[position] == 1
```

```
    def display(self):
        for row in self.grid:
            print(" ".join(map(str, row)))
```

5.2 AI-Agent Implementation

Listing 2: AI-Agent Implementation

```
class AIAgent:
    def __init__(self, environment):
        self.environment = environment
        self.position = (0, 0)
        self.state = "idle"

    def perceive(self):
        # Perceive the environment
        pass

    def decide(self):
        # Make decisions based on perceptions
        pass

    def act(self):
        # Execute actions
        pass

    def update(self):
        self.perceive()
        self.decide()
        self.act()
```

5.3 Pathfinding using A* Algorithm

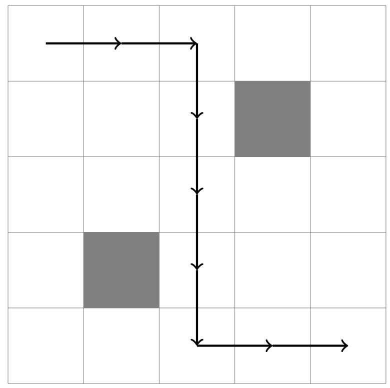
Listing 3: Pathfinding using A* Algorithm

```
import heapq

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

    open_set = []
    heapq.heappush(open_set, (0, start))
    came_from = {}
    g_score = {start: 0}
```

```
    f_score = {start: heuristic(start, goal)}

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

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.reverse()
            return path

        for neighbor in get_neighbors(current, environment):
            tentative_g_score = g_score[current] + 1
            if tentative_g_score < g_score.get(neighbor, float('inf')):
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor, goal
                heapq.heappush(open_set, (f_score[neighbor], neighbor))
    return []

def get_neighbors(position, environment):
    neighbors = []
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for direction in directions:
        neighbor = (position[0] + direction[0], position[1] + direction[1])
        if 0 <= neighbor[0] < environment.grid_size and 0 <= neighbor[1] < enviro
            if not environment.is_obstacle(neighbor):
                neighbors.append(neighbor)
    return neighbors
```

6 Results

The AI-Agent was tested in a grid-based environment with various obstacles. The A* algorithm successfully found the optimal path from the start to the goal position. The AI-Agent's behavior was managed using a finite state machine, allowing it to transition between idle, moving, and interacting states effectively.



7 Conclusion

The implementation of an AI-driven game character demonstrates the application of basic AI techniques such as pathfinding, finite state machines, and behavior trees. These techniques enable the AI-Agent to interact with the game environment and players in a meaningful way, providing a foundation for more complex AI behavior in games.