TOP-UP BACHELOR OF SCIENCE HONOURS INFORMATICS

Individual report on software implementation - CW3

Module Code: KOL304CR

Module: Games and AI

Module Leader: Lena Erbs

Name: Tania

Github Link: https://github.com/taneje02/Games-and-AI---CW3.git

Introduction

The game features in multiplayer Pac-Man scenario in which two players navigate a complex maze while evading pursuit by AI-controlled ghosts. A key gameplay requirement is for the ghosts to exhibit intelligent chasing behavior, making the game dynamic, responsive, and engaging. To achieve this, the A* pathfinding algorithm is applied. This method enables each ghost to calculate the shortest path to the nearest player in real time, allowing for adaptive and strategic movement. As a result, the AI enriches gameplay by introducing challenging enemy behavior that continuously responds to player actions.

The A* algorithm is ideal for grid-based pathfinding due to its accuracy and computational efficiency. This report implements a real-time AI system using the A* pathfinding algorithm within a Pac-Man-inspired multiplayer game and also explains the implementation of the AI, evaluates its effectiveness, and reflects on potential adaptations for multiplayer networked gameplay.

Implementation

The A* algorithm is employed to allow the ghosts to intelligently pursue the players within the maze. This is achieved using a priority queue to manage the nodes explored during the search process. The `astar` function calculates the shortest path between a ghost and the closest player, taking into account the layout and constraints of the maze grid. Once the path is determined, the ghost follows it step by step, continuously updating its position as it moves toward its target.

Here is the Key implementation details.

Maze Representation
The maze is defined as a 2D grid of 1s (walls) and 0s (walkable paths). Each entity has its position tracked as (x, y) grid coordinates.

```python
MAZE = [[int(cell) for cell in row] for row in raw_maze]
```

Each game object (players and ghosts) has its position tracked as `(x, y)` grid coordinates, which directly correlate to screen positions using a constant `TILE_SIZE`.

A* Pathfinding Algorithm

The `astar(start, goal)` function implements A* using a priority queue. It calculates a path from a ghost to the target player, using travel cost + Manhattan heuristic.

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

def astar(start, goal):
    heapq.heappush(queue, (0, start))
    ...
    priority = new_cost + heuristic(goal, next_node)
    heapq.heappush(queue, (priority, next_node))
```

Manhattan distance is used as the heuristic due to its efficiency on grid-based movement without diagonals.

Ghost Behavior

Each ghost identifies the closest player and recalculates its optimal paths

```python
distances = [(heuristic((self.x, self.y), (p.x, p.y)), p) for p in players]
target = min(distances, key=lambda x: x[0])[1]
path = astar((self.x, self.y), (target.x, target.y))
```

Ghosts move less frequently than players (every 3 frames) for balance. This ensures the game remains playable while still maintaining pressure on the players.

Collision Detection

If a ghost's coordinates match a player's, the game ends immediately.

```python
if ghost.x == player.x and ghost.y == player.y:
    return True
```

Key Design Decisions

- **Speed Balancing:** Players move every 2 frames, while ghosts move every frame, making ghosts slightly faster.
- **Dynamic Targeting:** Ghosts recalculate their target path every frame to track players even as they move.
- **Game Over Mechanic:** The game ends and shows a “Game Over!” message if any ghost catches a player.

Testing and Evaluation

The implementation was tested across multiple maze layouts and randomized player/ghost spawn positions to validate the correctness and performance of the A* pathfinding system.

- **Path accuracy:** Ensure that ghosts always reach the player using the shortest possible valid route.
- **Responsiveness:** Verify that ghost behavior updates dynamically in response to moving players.
- **Collision correctness:** Ensure the game ends instantly when a ghost catches a player.

Strengths

1. Optimal Pathfinding: A* guarantees the **shortest path** in a grid if the heuristic (Manhattan distance) is admissible, which it is for this maze setup. Unlike random or greedy methods, it doesn't just move closer, it intelligently avoids dead ends and detours.

2. **Dynamic Ghost Behavior:** Ghosts do not follow hardcoded routes or scripted patterns. Their behavior adjusts based on player movement in real time, which increases the challenge and realism of the game.

3. **Real-Time Multi-Agent Support:** Even with multiple players and ghosts, the system maintains responsiveness. Each ghost can independently decide which player to target based on current positions, making interactions emergent and interesting.

Weaknesses

1. **Predictability:** A major trade-off of always choosing the optimal path is **lack of unpredictability**. Players can learn ghost behavior and exploit it (e.g., leading ghosts into loops or safe zones).

2. **Scalability Limitations:** On large mazes or with many ghosts, recalculating A* paths every few frames can become computationally expensive. Python (especially with Pygame) is not optimized for high-performance pathfinding, and CPU usage may rise.

3. **No Path Caching:** A* is recomputed from scratch for every ghost every few frames. In cases where players haven't moved much, reusing part of the previous path or implementing incremental **A*** could reduce computation.

Gameplay Impact

The AI significantly enhances gameplay tension. Players are constantly pursued by ghosts who intelligently adapt to movements. This simulates realistic and engaging chase behavior with minimal computational overhead.

Reflection on Network Principles

In a networked multiplayer game, applying A* pathfinding AI introduces greater complexity than in a local case. While A* remains a valid and efficient algorithm for finding the shortest route in maze-like situations. A* pathfinding in a networked game must be able to handle problems involving latency and synchronization. Since A* applies current, real-time data, any delay in transmitting player positions can cause ghosts to chase outdated targets. This impacts gameplay correctness and player experience as well. For consistency, A* would have to be run on the server side and be the absolute authority for ghost movement so as to eliminate variation between players. However, this incurs server load and network traffic, especially with multiple ghosts constantly redistributing paths. Client-server dessynchronization may result in each player receiving a different ghost location. To minimize these issues, developers can use methods like limiting path updates, path caching, and client-side interpolation to achieve smoother ghost movement. A* provides optimal pathfinding, but it must be applied in a network environment with proper design for performance, fairness, and consistent AI behavior across all connected clients.

Conclusion

The implementation of A* pathfinding in a multiplayer Pacman made the ghost AI more enhanced, with the enemy AI becoming more difficult and smarter. While there is potential for improvement, like the ghost speed and strategic decision-making, the present setup has already managed to make the game more challenging. Future expansions could include diverse ghost strategies, power-ups, and networked multiplayer support.

Appendix (Full Source Code)

Full source code is provided in the multiplayer pac.py under the given github repository.

Path- multiplayer pac.py