

AI components

AI Components Documentation - Tài liệu Hệ thống AI

Tổng quan (Overview)

Hệ thống AI trong Pygame Pacman được thiết kế modular với 4 components chính:

- **AIPlayer**: Lớp chính điều khiển AI player
 - **AIState**: Quản lý trạng thái và dữ liệu AI
 - **BehaviorManager**: Quản lý các hành vi AI khác nhau
 - **PathfindingManager**: Cung cấp các thuật toán tìm đường
 - **DecisionMaker**: Phân tích tình huống và đưa ra quyết định
-

AIPlayer Class

Mô tả

Lớp chính kế thừa từ `PlayerBase`, điều khiển AI player với các thuật toán thông minh.

Khởi tạo

```
AIPlayer(player_id, start_x, start_y, sprite_manager,  
ai_type="simple_bfs")
```

Tính năng chính

1. Hệ thống chuyển động thông minh

- **Early direction change**: Cho phép đổi hướng sớm ($\text{progress} \leq 0.3$)
- **Stuck detection**: Phát hiện kẹt sau 30 frames
- **Random escape**: Thoát khỏi tình huống kẹt bằng hướng ngẫu nhiên



2. Power-up Management

```
# Khi ăn power pellet
if points == POWER_PELLET_POINTS:
    self.is_invincible = True
    self.power_timer = self.power_up_duration
```

3. Debug Rendering

- **Path visualization**: Hiển thị đường đi AI bằng đường màu xanh
- **Algorithm display**: Hiển thị tên thuật toán đang dùng
- **Power timer**: Hiển thị thời gian còn lại của power-up
- **Grid position**: Hiển thị vị trí grid và pixel

4. Visual Effects

- **Power glow**: Hiệu ứng sáng khi có power-up
 -  **Vàng**: Thời gian > 60 frames
 -  **Đỏ**: Thời gian ≤ 60 frames
- **Invincibility blink**: Nhấp nháy khi bất tử

Quy trình Update

1. **Update base states**: Invincibility, animation
2. **Update AI state**: Counters, timers
3. **Handle power timer**: Giảm dần power_timer
4. **Check stuck condition**: Phát hiện và xử lý kẹt
5. **Make decisions**: Phân tích tình huống mỗi 8 frames
6. **Handle movement**: Xử lý di chuyển thực tế



AISState Class



Mô tả

Quản lý trạng thái, dữ liệu và thống kê của AI player.



Dữ liệu chính

```
class AISState:
    def __init__(self, ai_type, start_x, start_y):
        self.ai_type = ai_type           # Loại thuật toán
        self.path = []                  # Đường đi hiện tại
        self.target = None               # Mục tiêu hiện tại
        self.last_position = (start_x, start_y)  # Vị trí trước đó
        self.decision_timer = 0          # Timer cho quyết định
```

```
self.stuck_counter = 0
self.recent_positions = []
```

```
# Đếm số frame bị kẹt
# Lịch sử vị trí gần đây
```



Thống kê AI

- **pellets_collected**: Số pellet đã ăn
- **power_pellets_eaten**: Số power pellet đã ăn
- **ghosts_eaten**: Số ma đã ăn
- **deaths**: Số lần chết
- **total_distance**: Tổng quãng đường đi



BehaviorManager Class



Mô tả

Quản lý và chuyển đổi giữa các hành vi AI khác nhau trong runtime.



Các hành vi có sẵn

1. Simple Algorithms (`simple.py`)

- **BFS**: Breadth-First Search - Tìm đường ngắn nhất
- **DFS**: Depth-First Search - Khám phá sâu
- **A***: A-Star - Tối ưu với heuristic
- **UCS**: Uniform Cost Search - Chi phí đồng nhất

2. Reflex Agent (`reflex_agent.py`)

- **Ghost prediction**: Dự đoán vị trí ma
- **Smart pellet collection**: Thu thập pellet thông minh
- **Dynamic escape**: Thoát hiểm thông minh

3. Smart Hunter (`smart_hunter.py`)

- **Aggressive hunting**: Săn ma khi có power-up
- **Strategic power pellet**: Sử dụng power pellet chiến lược
- **Risk assessment**: Đánh giá rủi ro



Chuyển đổi hành vi

```
def change_algorithm(self, new_algorithm):
    if new_algorithm in self.behavior_manager.behaviors:
```

```
self.ai_state.ai_type = new_algorithm
```



PathfindingManager Class



Mô tả

Cung cấp các thuật toán tìm đường từ cơ bản đến nâng cao.



Thuật toán có sẵn

1. BFS (Breadth-First Search)

- ✓ **Ưu điểm:** Tìm đường ngắn nhất
- ✗ **Nhược điểm:** Chậm với map lớn
- 🎯 **Sử dụng:** Tìm pellet gần nhất

2. DFS (Depth-First Search)

- ✓ **Ưu điểm:** Nhanh, tiết kiệm bộ nhớ
- ✗ **Nhược điểm:** Không tối ưu
- 🎯 **Sử dụng:** Khám phá map

3. A* (A-Star)

- ✓ **Ưu điểm:** Tối ưu + nhanh
- ✗ **Nhược điểm:** Phức tạp hơn
- 🎯 **Sử dụng:** Tìm đường tối ưu

4. UCS (Uniform Cost Search)

- ✓ **Ưu điểm:** Tối ưu với trọng số
- ✗ **Nhược điểm:** Chậm hơn A*
- 🎯 **Sử dụng:** Terrain có chi phí khác nhau



Heuristic Functions

```
def manhattan_distance(pos1, pos2):  
    return abs(pos1[0] - pos2[0]) + abs(pos1[1] - pos2[1])
```



DecisionMaker Class



Mô tả

Phân tích tình huống game và đưa ra quyết định tối ưu cho AI.

Phân tích tình huống

```
def analyze_situation(self, maze, player_position, ghosts_positions):
    return {
        'nearest_pellet': self.find_nearest_pellet(maze),
        'nearest_power_pellet': self.find_nearest_power_pellet(maze),
        'ghost_threats': self.assess_ghost_threats(ghosts_positions),
        'safe_directions': self.find_safe_directions(maze,
ghosts_positions),
        'game_phase': self.determine_game_phase(maze)
    }
```

Các pha game

1. **Early Game**: Thu thập pellet an toàn
2. **Mid Game**: Cân bằng risk/reward
3. **End Game**: Tích cực và quyết đoán

Cách sử dụng

1. Tạo AI Player

```
ai_player = AIPlayer("ai1", 1, 1, sprite_manager, "simple_bfs")
```

2. Chuyển đổi thuật toán

```
ai_player.change_algorithm("reflex_agent")
available = ai_player.get_available_algorithms()
current = ai_player.get_current_algorithm()
```

3. Update trong game loop

```
ai_player.update(maze, player_position, ghosts_positions)
```

4. Render với debug

```
ai_player.render(screen, debug=True)
```



📊 Performance Tips

🚀 Tối ưu hóa

1. ****Giảm tần suất quyết định****: Tăng `decision_timer`
2. ****Cache đường đi****: Lưu lại path đã tính
3. ****Limit search depth****: Giới hạn độ sâu tìm kiếm
4. ****Precompute heuristics****: Tính trước heuristic

🇧🇷 Monitoring

```python

# Kiểm tra performance

print(f"Decisions per second: {decisions / elapsed\_time}")

print(f"Path calculation time: {path\_time}ms")

print(f"Memory usage: {len(ai\_state.recent\_positions)}")

## 🐛 Debug và Testing

### 🔍 Debug Mode

# Hiển thị thông tin debug

ai\_player.render(screen, debug=True)

### 📋 Debug Info hiển thị:

- **Red rectangle**: Grid position
- **Blue circle**: Pixel position
- **Green line**: Planned path
- **Text**: Algorithm name & power timer
- **Glow effect**: Power-up status

### 🧪 Testing

# Test các thuật toán






for algorithm in ai\_player.get\_available\_algorithms():

```
ai_player.change_algorithm(algorithm)
test_performance(ai_player)
```



## Kết luận

Hệ thống AI được thiết kế:

-  **Modular**: Dễ mở rộng và tùy chỉnh
-  **Intelligent**: Nhiều thuật toán thông minh
-  **Flexible**: Chuyển đổi hành vi runtime
-  **Debuggable**: Tools debug mạnh mẽ
-  **Scalable**: Có thể thêm AI mới dễ dàng

Hệ thống này cho phép tạo ra AI players với nhiều mức độ thông minh khác nhau, từ cơ bản đến nâng cao, phù hợp cho việc học tập và nghiên cứu AI trong game development.