

SOUTENANCE PROJET IA



SOMMAIRE

1- PROJECT GOAL

2- FSM

3- A*

4- BT



2-FSM

POUR LE PREMIER ENNEMI, NOUS AVONS UTILISÉ LE FSM.
C'EST L'ENNEMI ORANGE.

```
void Enemy::patrol(Grid& grid) {  
  
    static int currentWaypoint = 0;  
    static sf::Vector2f waypoints[4] = /*{ sf::Vector2f(100, 300), sf::Vector2f(500, 100), sf::Vector2f(100, 3  
        sf::Vector2f(initialPos.x - 150, initialPos.y + 150), sf::Vector2f(initialPos.x + 150, initialPos.y -  
    sf::Vector2f target = waypoints[currentWaypoint];  
    sf::Vector2f direction = target - pos;  
    float distance = std::sqrt(direction.x * direction.x + direction.y * direction.y);  
    if (grid.getCell(pos.x / CELL_SIZE, pos.y / CELL_SIZE).walkable == true) {  
        if (distance < 5.0f) {  
            currentWaypoint = (currentWaypoint + 1) % 4;  
        }  
        else {  
            direction /= distance;  
            pos += direction * 1.7f;  
            e_direction = direction;  
        }  
        //shape.move(direction * SPEED);  
    }  
}
```

```
void Enemy::chase(Vector2f playerPos) {  
    Vector2f direction = playerPos - pos;  
    float distance = sqrt(direction.x * direction.x + direction.y * direction.y);  
  
    if (distance > 0) {  
        direction /= distance;  
        pos += direction * 2.0f;  
        e_direction = direction;  
    }  
  
    shape.move(direction*SPEED);  
}
```

```
void Enemy::update(float deltaTime, Grid& grid, std::vector<Entity*> players, sf::Vector2f playerPos) {  
    shape.setPosition(pos);  
    switch (currentState) {  
        case PATROL: {  
  
            patrol(grid);  
            if (detectPlayer(players[0]->pos)) {  
                lastPlayerPosition = players[0]->pos;  
                currentState = CHASE;  
            }  
            break;  
        }  
  
        case CHASE: {  
  
            chase(players[0]->pos);  
            if (!detectPlayer(players[0]->pos)) {  
                currentState = PATROL;  
            }  
            break;  
        }  
  
        case SEARCH:  
            search(lastPlayerPosition, deltaTime);  
            break;  
    }  
}
```

3-A*

POUR LE DEUXIÈME ENNEMI, NOUS AVONS UTILISÉ LE FSM.
C'EST L'ENNEMI BLEU.

```
void A_Enemy::Path(Vector2i start, Vector2i end, Pathfinding& path, Grid& grid) {  
    if (start != previousStart || end != previousEnd) {  
        currentPath.clear();  
        currentPathIndex = 0;  
        previousStart = start;  
        previousEnd = end;  
    }  
  
    if (currentPath.empty()) {  
        currentPath = path.findPath(grid, start, end);  
        currentPathIndex = 0;  
        reversePath = false;  
    }  
  
    if (!currentPath.empty() && currentPathIndex < currentPath.size()) {  
        Vector2i target = currentPath[currentPathIndex];  
        Vector2f targetPos(target.x * CELL_SIZE, target.y * CELL_SIZE);  
        Vector2f direction = targetPos - pos;  
  
        float distance = sqrt(direction.x * direction.x + direction.y * direction.y);  
  
        if (distance < 5.0f) {  
            currentPathIndex++;  
        }  
        else {  
            direction /= distance;  
            pos += direction * 2.0f;  
        }  
    }  
    shape.setPosition(pos);  
}
```

4- BT

POUR LE TROISIÈME ENNEMI, NOUS AVONS UTILISÉ LE BT.
C'EST L'ENNEMI VERT.

```
SelectorNode::SelectorNode(Blackboard& blackboard) {
    std::unique_ptr<SequenceNode> sequence = std::make_unique<SequenceNode>();
    sequence->AddChild(std::make_unique<ConditionNode>(blackboard, "getAway", 0));
    sequence->AddChild(std::make_unique<ConditionNode>(blackboard, "PlayerDetected", 0));
    sequence->AddChild(std::make_unique<ActionNode>("movement"));
    std::unique_ptr<SequenceNode> sequence1 = std::make_unique<SequenceNode>();
    sequence1->AddChild(std::make_unique<ConditionNode>(blackboard, "PlayerDetected", 1));
    sequence1->AddChild(std::make_unique<ActionNode>("chase"));
    sequence1->AddChild(std::make_unique<ConditionNode>(blackboard, "Cooldown", 1));
    sequence1->AddChild(std::make_unique<ActionNode>("shoot"));
    std::unique_ptr<SequenceNode> sequence2 = std::make_unique<SequenceNode>();
    sequence2->AddChild(std::make_unique<ConditionNode>(blackboard, "getAway", 1));
    sequence2->AddChild(std::make_unique<ConditionNode>(blackboard, "wallCollision", 0));
    sequence2->AddChild(std::make_unique<ActionNode>("getAway"));
    std::unique_ptr<SequenceNode> sequence3 = std::make_unique<SequenceNode>();
    sequence3->AddChild(std::make_unique<ConditionNode>(blackboard, "wallCollision", 1));
    sequence3->AddChild(std::make_unique<ActionNode>("wallCollision"));
    AddChild(std::move(sequence));
    AddChild(std::move(sequence1));
    AddChild(std::move(sequence2));
    AddChild(std::move(sequence3));
}
```

MERCI DE
NOUS
AVOIR
ECOUTÉ

