CSE 221 - Object Oriented Programming

Chess Project

Table of Contents

Project Overview

Description

Implement a custom chess game with configurable pieces and portals. The game extends traditional chess with fantasy elements like teleportation portals and custom movement patterns.

Key Features

- Traditional chess pieces with standard rules
- Configurable board size and turn limits
- Portal system for piece teleportation
- Custom movement patterns
- JSON-based configuration

Class Architecture

Core Classes and Interactions

1. GameManager

Responsibilities:

- Game state management
- Turn processing
- Win/loss condition checking
- Player management

2. ChessBoard

Responsibilities:

- Board state representation
- Piece placement and movement
- Capture handling

3. MoveValidator

Responsibilities:

- Move validation
- Path checking
- Portal usage validation

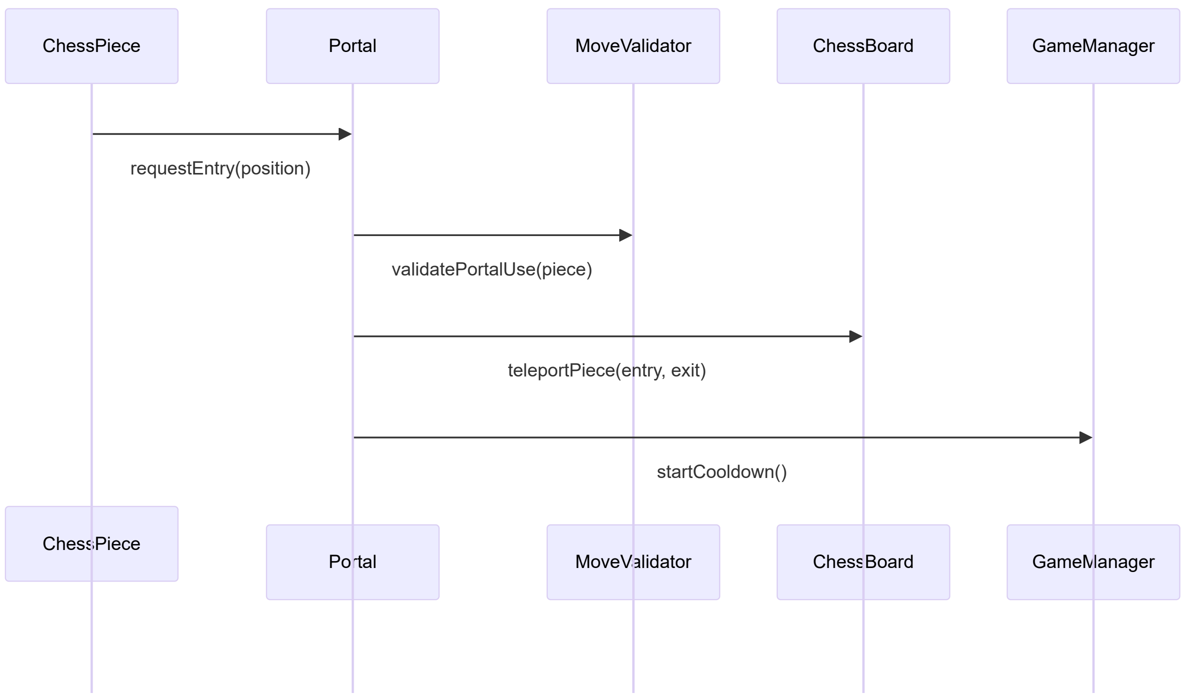
4. Portal System

Responsibilities:

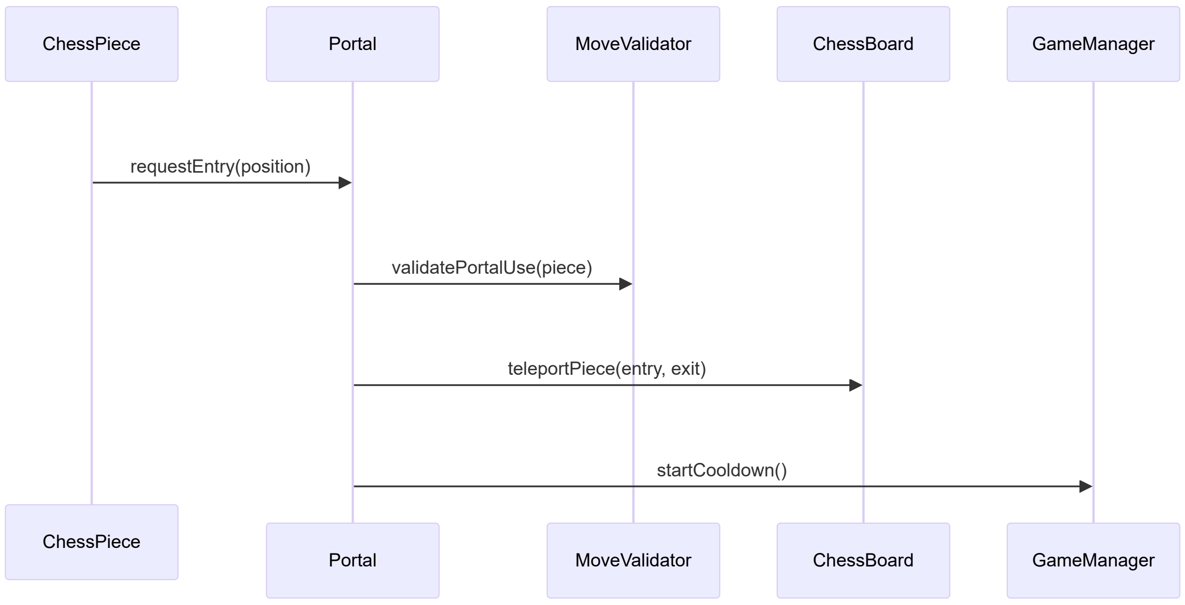
- Teleportation logic
- Cooldown management
- Direction preservation

Class Interaction Flow

1. **Move Processing:**



1. **Portal Usage:**



Game Mechanics

Movement Rules

1. **Standard Pieces**
   - Follow traditional chess rules
   - Additional portal interaction capabilities
2. **Portal Mechanics**
   - Entry and exit points
   - Direction preservation options
   - Color-specific restrictions
   - Cooldown periods

Turn Processing

1. Player selects piece
2. System validates possible moves
3. Player chooses destination
4. System processes move:
   - Regular movement
   - Capture handling
   - Portal usage
   - State updates

Configuration System

File Structure

```
{
  "game_settings": {
    "name": "string",
    "board_size": "integer",
    "turn_limit": "integer"
  },
  "pieces": [...],
  "portals": [...]
}
```

Validation Rules

1. Board boundaries
2. Piece counts
3. Portal placement
4. Movement patterns

Implementation Requirements

Required Features

1. Complete move validation
2. Portal system implementation
3. Game state persistence
4. Configuration loading
5. Win/loss detection

Code Quality Standards

1. Comprehensive documentation
2. Unit test coverage
3. Error handling
4. Memory management
5. Code style consistency

Building and Testing

Prerequisites

- nlohmann/json library
- Make build system

Build Commands

```
make deps      # Install dependencies
make          # Build project
make run      # Run with default config
make clean    # Clean build files
```

Testing Strategy

1. Unit Tests
    - Piece movement
    - Portal mechanics
    - Game state
    - Configuration parsing
2. Integration Tests
    - Complete game flows
    - Portal interactions
    - State management

Deliverables

1. Source code
2. Configuration files
3. Test suite
4. Documentation
5. Build scripts

Submission Guidelines

1. Clean, documented code
2. Complete test suite
3. README with build instructions
4. Sample configurations
5. Design documentation