Docker Adventure Game - Complete Solution

Directory Structure

```
c:\src\dockergame\
├── shared/                 # Shared volume directory
│   ├── game.log            # Game output and debug log
│   └── save_data/          # Future save game data
├── Dockerfile
├── docker-compose.yml
├── requirements.txt
├── src/
│   ├── main.py             # Main game loop
│   ├── game_engine.py      # Core game engine
│   ├── world.py            # World and room definitions
│   ├── player.py           # Player class
│   ├── items.py            # Game items
│   ├── commands.py         # Command parser and handlers
│   ├── ollama_client.py    # Ollama integration
│   └── utils.py            # Utility functions
└── README.md
```

Files Content

Dockerfile

dockerfile

```
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements and install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy source code
COPY src/ ./src/

# Create shared directory inside container
RUN mkdir -p /app/shared

# Set environment variables
ENV PYTHONUNBUFFERED=1
ENV PYTHONPATH=/app

# Keep container running
CMD ["python", "src/main.py"]
```

docker-compose.yml

yaml

```yaml
version: '3.8'

services:
  adventure-game:
    build: .
    container_name: docker-adventure-game
    volumes:
      - c:\src\dockergame\shared:/app/shared
    environment:
      - OLLAMA_HOST=host.docker.internal:11434
      - GAME_LOG_FILE=/app/shared/game.log
    stdin_open: true
    tty: true
    restart: unless-stopped
    networks:
      - game-network

networks:
  game-network:
    driver: bridge
```

requirements.txt

```
requests>=2.31.0
colorama>=0.4.6
python-dateutil>=2.8.2
```

src/main.py

python

```python
#!/usr/bin/env python3
"""
Main entry point for the Docker Adventure Game
"""

import sys
import signal
import time
from pathlib import Path

from game_engine import GameEngine
from utils import Logger

def signal_handler(sig, frame):
    """Handle graceful shutdown"""
    print("\n\nGame shutting down gracefully...")
    sys.exit(0)

def main():
    """Main game loop"""
    # Set up signal handler
    signal.signal(signal.SIGINT, signal_handler)
    signal.signal(signal.SIGTERM, signal_handler)

    # Initialize logger
    logger = Logger()
    logger.log("=== Docker Adventure Game Starting ===")
    logger.log("Game initialized successfully")

    try:
        # Initialize game engine
        game = GameEngine(logger)

        # Welcome message
        print("\n" + "="*50)
        print("    🏰 WELCOME TO DOCKER ADVENTURE GAME 🏰")
        print("="*50)
        print("Type 'help' for commands or 'quit' to exit")
        print("Commands support abbreviations (e.g., 'm n' for 'move north')")
        print("-"*50)

        # Main game loop
        while True:
```

```
            try:
                # Get user input
                user_input = input("\n> ").strip()

                if not user_input:
                    continue

                # Log user input
                logger.log(f"User input: {user_input}")

                # Process command
                result = game.process_command(user_input)

                # Handle special commands
                if result == "QUIT":
                    print("\nThanks for playing! Goodbye! 👋")
                    logger.log("Game ended by user")
                    break
                elif result == "HELP":
                    game.show_help()
                else:
                    # Print command result
                    if result:
                        print(result)
                        logger.log(f"Command result: {result}")

            except KeyboardInterrupt:
                print("\n\nUse 'quit' to exit gracefully.")
                continue
            except Exception as e:
                error_msg = f"Error processing command: {str(e)}"
                print(f"❌ {error_msg}")
                logger.log(f"ERROR: {error_msg}")

    except Exception as e:
        error_msg = f"Fatal error: {str(e)}"
        print(f"💥 {error_msg}")
        logger.log(f"FATAL: {error_msg}")
        return 1

    logger.log("=== Docker Adventure Game Ended ===")
    return 0

if __name__ == "__main__":
```

```
    exit_code = main()
    sys.exit(exit_code)
```

src/game_engine.py

python

```
"""
Core game engine for the adventure game
"""

from player import Player
from world import World
from commands import CommandParser
from ollama_client import OllamaClient

class GameEngine:
    def __init__(self, logger):
        self.logger = logger
        self.player = Player()
        self.world = World()
        self.command_parser = CommandParser()
        self.ollama_client = OllamaClient(logger)

        # Place player in starting room
        self.player.current_room = "forest_entrance"

        # Show initial room description
        self._show_room_description()

    def process_command(self, user_input):
        """Process user command and return result"""
        if user_input.lower() in ['quit', 'exit', 'q']:
            return "QUIT"
        elif user_input.lower() in ['help', 'h', '?']:
            return "HELP"

        # Parse command
        command, subcommand, args = self.command_parser.parse(user_input)

        if not command:
            return "❓ I don't understand that command. Type 'help' for available commands."

        # Execute command
        return self._execute_command(command, subcommand, args)

    def _execute_command(self, command, subcommand, args):
        """Execute parsed command"""
        try:
            if command == "look":
```

```
                return self._handle_look(subcommand)
            elif command == "move":
                return self._handle_move(subcommand)
            elif command == "grab":
                return self._handle_grab(args)
            elif command == "inventory":
                return self._handle_inventory()
            elif command == "fight":
                return self._handle_fight(args)
            else:
                return f"❓ Unknown command: {command}"

        except Exception as e:
            self.logger.log(f"Command execution error: {str(e)}")
            return f"❌ Error executing command: {str(e)}"

    def _handle_look(self, direction=None):
        """Handle look command"""
        current_room = self.world.get_room(self.player.current_room)

        if not direction:
            # Look around current room
            result = f"🏠 {current_room['name']}\n"
            result += f"📝 {current_room['description']}\n"

            # Show items
            if current_room.get('items'):
                result += f"📦 Items here: {', '.join(current_room['items'])}\n"

            # Show exits
            exits = [direction for direction in ['north', 'south', 'east', 'west']
                     if current_room.get('exits', {}).get(direction)]
            if exits:
                result += f"🚪 Exits: {', '.join(exits)}"
            else:
                result += "🚪 No obvious exits"

            return result
        else:
            # Look in specific direction
            exit_room = current_room.get('exits', {}).get(direction)
            if exit_room:
                target_room = self.world.get_room(exit_room)
                return f"👀 To the {direction}: {target_room['name']} - {target_room.get('shor
```

```
            else:
                return f"👀 You see nothing interesting to the {direction}."

    def _handle_move(self, direction):
        """Handle move command"""
        if not direction:
            return "🚶 Move where? Specify a direction (north, south, east, west)"

        current_room = self.world.get_room(self.player.current_room)
        exit_room = current_room.get('exits', {}).get(direction)

        if not exit_room:
            return f"🚫 You can't go {direction} from here."

        # Move player
        self.player.current_room = exit_room
        self.logger.log(f"Player moved {direction} to {exit_room}")

        # Show new room description
        return self._show_room_description()

    def _handle_grab(self, item_name):
        """Handle grab command"""
        if not item_name:
            return "✊ Grab what? Specify an item name."

        current_room = self.world.get_room(self.player.current_room)
        room_items = current_room.get('items', [])

        # Find item (case insensitive)
        item_to_grab = None
        for item in room_items:
            if item.lower() == item_name.lower():
                item_to_grab = item
                break

        if not item_to_grab:
            return f"📦 There's no '{item_name}' here to grab."

        # Add to inventory and remove from room
        self.player.inventory.append(item_to_grab)
        room_items.remove(item_to_grab)

        self.logger.log(f"Player grabbed: {item_to_grab}")
```

```
        return f"✅ You grabbed the {item_to_grab}!"

    def _handle_inventory(self):
        """Handle inventory command"""
        if not self.player.inventory:
            return "🎒 Your inventory is empty."

        return f"🎒 Inventory: {', '.join(self.player.inventory)}"

    def _handle_fight(self, target):
        """Handle fight command with AI assistance"""
        if not target:
            return "⚔️ Fight what? You need to specify a target."

        current_room = self.world.get_room(self.player.current_room)

        # Check if there are enemies in the room
        enemies = current_room.get('enemies', [])
        if not enemies:
            return "⚔️ There's nothing to fight here."

        # Use AI to generate fight scenario
        fight_prompt = f"""
        The player is fighting a {target} in {current_room['name']}.
        The room description: {current_room['description']}
        Player inventory: {', '.join(self.player.inventory) if self.player.inventory else 'empt

        Generate a short, exciting fight outcome (2-3 sentences).
        Make it adventurous but not too violent.
        """

        ai_response = self.ollama_client.generate_response(fight_prompt)

        if ai_response:
            self.logger.log(f"Fight with {target}: {ai_response}")
            return f"⚔️ {ai_response}"
        else:
            return f"⚔️ You engage the {target} in combat! The battle is fierce but you emerge

    def _show_room_description(self):
        """Show current room description"""
        return self._handle_look()

    def show_help(self):
```

```
        """Show help information"""
        help_text = """
🎮 GAME COMMANDS:
═══════════════════════════════════════════════════

🎈 LOOK (l)              - Look around current room
   look <direction>      - Look in specific direction (n/s/e/w)

🚶 MOVE (m)              - Move in a direction
   move <direction>      - Move north/south/east/west (n/s/e/w)

✊ GRAB (g)              - Pick up an item
   grab <item>           - Grab specific item

🎒 INVENTORY (i)         - Show your inventory

⚔️ FIGHT (f)             - Fight an enemy
   fight <enemy>         - Fight specific enemy

❓ HELP (h)              - Show this help
🚪 QUIT (q)              - Exit game

💡 TIP: You can use abbreviations!
   'm n' = 'move north', 'l e' = 'look east', etc.

═══════════════════════════════════════════════════
        """
        print(help_text)
```

src/world.py

python

```
"""
World definition and room management
"""

class World:
    def __init__(self):
        self.rooms = {
            "forest_entrance": {
                "name": "Forest Entrance",
                "description": "You stand at the edge of a mysterious forest. Ancient trees tow
                "short_desc": "The entrance to a mysterious forest",
                "exits": {
                    "north": "forest_path",
                    "east": "old_well"
                },
                "items": ["stick", "stone"],
                "enemies": []
            },
            "forest_path": {
                "name": "Forest Path",
                "description": "A winding path leads deeper into the forest. You hear the dista
                "short_desc": "A winding forest path",
                "exits": {
                    "south": "forest_entrance",
                    "north": "forest_clearing",
                    "west": "dark_cave"
                },
                "items": ["berries"],
                "enemies": ["forest sprite"]
            },
            "old_well": {
                "name": "Old Well",
                "description": "An ancient stone well sits in a small clearing. Moss covers its
                "short_desc": "An ancient stone well",
                "exits": {
                    "west": "forest_entrance",
                    "north": "forest_clearing"
                },
                "items": ["rope", "coin"],
                "enemies": []
            },
            "forest_clearing": {
                "name": "Forest Clearing",
```

```
            "description": "A beautiful clearing opens up before you, bathed in golden sunl
            "short_desc": "A peaceful forest clearing",
            "exits": {
                "south": "forest_path",
                "east": "old_well",
                "west": "dark_cave"
            },
            "items": ["flowers", "crystal"],
            "enemies": []
        },
        "dark_cave": {
            "name": "Dark Cave",
            "description": "The cave entrance yawns before you like a hungry mouth. Cool, d
            "short_desc": "A dark, mysterious cave",
            "exits": {
                "east": "forest_path",
                "south": "forest_clearing"
            },
            "items": ["torch", "gem"],
            "enemies": ["cave troll"]
        }
    }

def get_room(self, room_id):
    """Get room by ID"""
    return self.rooms.get(room_id, {})

def get_all_rooms(self):
    """Get all rooms"""
    return self.rooms
```

src/player.py

python

```python
"""
Player class definition
"""

class Player:
    def __init__(self):
        self.name = "Adventurer"
        self.current_room = None
        self.inventory = []
        self.health = 100
        self.experience = 0
        self.level = 1

    def add_item(self, item):
        """Add item to inventory"""
        self.inventory.append(item)

    def remove_item(self, item):
        """Remove item from inventory"""
        if item in self.inventory:
            self.inventory.remove(item)
            return True
        return False

    def has_item(self, item):
        """Check if player has item"""
        return item in self.inventory

    def get_status(self):
        """Get player status"""
        return {
            "name": self.name,
            "health": self.health,
            "level": self.level,
            "experience": self.experience,
            "inventory_count": len(self.inventory)
        }
```

src/commands.py

python

```
"""
Command parsing and handling
"""

class CommandParser:
    def __init__(self):
        # Command mappings with abbreviations
        self.commands = {
            'look': ['look', 'l'],
            'move': ['move', 'm', 'go'],
            'grab': ['grab', 'g', 'take', 'get'],
            'inventory': ['inventory', 'i', 'inv'],
            'fight': ['fight', 'f', 'attack', 'battle']
        }

        # Direction mappings
        self.directions = {
            'north': ['north', 'n'],
            'south': ['south', 's'],
            'east': ['east', 'e'],
            'west': ['west', 'w']
        }

    def parse(self, user_input):
        """Parse user input into command, subcommand, and arguments"""
        if not user_input:
            return None, None, None

        parts = user_input.lower().strip().split()
        if not parts:
            return None, None, None

        # Find command
        command = self._find_command(parts[0])
        if not command:
            return None, None, None

        # Parse subcommand and arguments
        subcommand = None
        args = None

        if len(parts) > 1:
            # Check if second part is a direction
```

```
            direction = self._find_direction(parts[1])
            if direction and command in ['look', 'move']:
                subcommand = direction
            else:
                # Treat as argument
                args = ' '.join(parts[1:])

        return command, subcommand, args

    def _find_command(self, input_cmd):
        """Find command from input (including abbreviations)"""
        for cmd, aliases in self.commands.items():
            if input_cmd in aliases:
                return cmd
        return None

    def _find_direction(self, input_dir):
        """Find direction from input (including abbreviations)"""
        for direction, aliases in self.directions.items():
            if input_dir in aliases:
                return direction
        return None
```

src/ollama_client.py

python

```python
"""
Ollama client for AI-powered responses
"""

import requests
import json
import os

class OllamaClient:
    def __init__(self, logger):
        self.logger = logger
        self.host = os.getenv('OLLAMA_HOST', 'localhost:11434')
        self.model = 'gemma2'
        self.base_url = f"http://{self.host}"

    def generate_response(self, prompt, max_tokens=150):
        """Generate AI response using Ollama"""
        try:
            url = f"{self.base_url}/api/generate"

            payload = {
                "model": self.model,
                "prompt": prompt,
                "stream": False,
                "options": {
                    "num_predict": max_tokens,
                    "temperature": 0.7
                }
            }

            self.logger.log(f"Sending request to Ollama: {prompt[:100]}...")

            response = requests.post(url, json=payload, timeout=30)

            if response.status_code == 200:
                result = response.json()
                ai_response = result.get('response', '').strip()
                self.logger.log(f"Ollama response: {ai_response[:100]}...")
                return ai_response
            else:
                self.logger.log(f"Ollama error {response.status_code}: {response.text}")
                return None
```

```
        except requests.exceptions.ConnectionError:
            self.logger.log("Cannot connect to Ollama service")
            return None
        except requests.exceptions.Timeout:
            self.logger.log("Ollama request timed out")
            return None
        except Exception as e:
            self.logger.log(f"Ollama client error: {str(e)}")
            return None

    def is_available(self):
        """Check if Ollama service is available"""
        try:
            response = requests.get(f"{self.base_url}/api/tags", timeout=5)
            return response.status_code == 200
        except:
            return False
```

src/utils.py

python

```
"""
Utility functions and helpers
"""

import os
from datetime import datetime
from pathlib import Path

class Logger:
    def __init__(self):
        self.log_file = os.getenv('GAME_LOG_FILE', '/app/shared/game.log')
        self.ensure_log_dir()

    def ensure_log_dir(self):
        """Ensure log directory exists"""
        log_path = Path(self.log_file)
        log_path.parent.mkdir(parents=True, exist_ok=True)

    def log(self, message):
        """Log message to file and optionally to console"""
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        log_entry = f"[{timestamp}] {message}"

        # Write to file
        try:
            with open(self.log_file, 'a', encoding='utf-8') as f:
                f.write(log_entry + '\n')
        except Exception as e:
            print(f"Logging error: {e}")

        # Also print debug info (optional)
        if os.getenv('DEBUG', '').lower() == 'true':
            print(f"DEBUG: {log_entry}")

def format_text(text, width=70):
    """Format text to specified width"""
    words = text.split()
    lines = []
    current_line = ""

    for word in words:
        if len(current_line + " " + word) <= width:
            if current_line:
```

```
                current_line += " " + word
            else:
                current_line = word
        else:
            if current_line:
                lines.append(current_line)
            current_line = word

    if current_line:
        lines.append(current_line)

    return '\n'.join(lines)
```

README.md

```
markdown

# Docker Adventure Game

A text-based adventure game running in Docker with AI-powered interactions using Ollama.

## Quick Start

1. **Prerequisites:**
   - Docker and Docker Compose installed
   - Ollama running on Windows 11 host with gemma2 model

2. **Setup:**
   ```bash
   cd c:\src\dockergame
   docker-compose up --build
```

3. **Play:**
   - The game will start automatically
   - Type commands like "look", "move north", "grab stick"
   - Use abbreviations: "m n" for "move north", "l" for "look"

Commands

- **look (l)** - Examine current room or direction
- **move (m) <direction>** - Move north/south/east/west
- **grab (g) <item>** - Pick up items
```

- **inventory (i)** - Show your items
- **fight (f) <enemy>** - Battle enemies (AI-powered)
- **help (h)** - Show help
- **quit (q)** - Exit game

Features

- ✅ Persistent logging to shared volume
- ✅ AI-powered fight scenarios via Ollama
- ✅ Expandable world and command system
- ✅ Command abbreviations support
- ✅ Container stays alive until stopped
- ✅ Comprehensive error handling

File Locations

- **Game logs:** `c:\src\dockergame\shared\game.log`
- **Source code:** `c:\src\dockergame\src\`
- **Configuration:** `docker-compose.yml`

Expanding the Game

1. **Add new rooms:** Edit `src/world.py`
2. **Add new commands:** Edit `src/commands.py` and `src/game_engine.py`
3. **Add new items:** Edit rooms in `src/world.py`
4. **Modify AI behavior:** Edit `src/ollama_client.py`

Troubleshooting

- Check logs: `c:\src\dockergame\shared\game.log`
- Restart container: `docker-compose restart`
- Rebuild: `docker-compose up --build`
- Ollama connection issues: Ensure Ollama is running on host

```
## Installation Instructions

1. **Create the directory structure:**
   ```cmd
   mkdir c:\src\dockergame
   cd c:\src\dockergame
   mkdir shared
   mkdir src
```

2. **Create all the files** as shown above in their respective directories
3. **Start Ollama** on your Windows 11 host:

   ```cmd
   ollama serve
   ollama pull gemma2
   ```

4. **Build and run the game:**

   ```cmd
   docker-compose up --build
   ```

5. **Play the game!** The container will stay running and you can interact with it directly.

Key Features Implemented

- ✅ **Docker container with volume binding**
- ✅ **Persistent logging to shared volume**
- ✅ **Command abbreviations (m n = move north)**
- ✅ **All requested commands: look, move, grab, inventory, fight**
- ✅ **Directional subcommands for look and move**
- ✅ **Container stays alive (doesn't exit)**
- ✅ **Easy to maintain and expand architecture**
- ✅ **Ollama integration for AI-powered interactions**
- ✅ **Comprehensive error handling and logging**

The game is fully modular - you can easily add new rooms in `world.py`, new commands in `commands.py`, and new features throughout the codebase!
   ```