NPC Colosseum Bot Programming Manual

DeepSeek

October 24, 2025

Create intelligent bots to battle in the arena!

Contents

1 Introduction

Welcome to the NPC Colosseum Bot Programming Guide! This document will help you understand how to create your own AI bots to battle in the arena. In NPC Colosseum, you can program bots that control characters during AI vs AI matches. Your bot will receive game state information and must return valid actions to compete against other bots.

2 Bot Function Structure

Your bot must be implemented as a Python function with the following signature:

```
def get_bot_action(current_game_state, game_archive):
    # Your bot logic here
    return "Valid Action String"
```

Listing 1: Bot Function Signature

2.1 Parameters

- `current_game_state`: A dictionary containing the current state of the game
- `game_archive`: A list containing the complete history of all turns played so far, including actions taken and their results

2.2 Return Value

Your function must return a string representing a valid action.

2.3 Class selection

You must also implement a class selection function `get_bot_class`

```
def get_bot_class():
    return class
```

Listing 2: Bot Function Signature

Class is a number between 0 and 2.

- 0: Warrior
- 1: Mage
- 2: Rogue

3 Game State Structure

The `current_game_state` dictionary contains:

```
{
    "player": {
        "hp": current_health_points,
        "max_hp": maximum_health_points,
        "skill_points": current_skill_points,
        "character_class": "Warrior"  # or "Mage" or "Rogue"
    },
    "opponent": {
        "hp": current_health_points,
        "max_hp": maximum_health_points,
        "skill_points": current_skill_points,
        "character_class": "Warrior"  # or "Mage" or "Rogue"
    },
    "inventory": {
        "Potion": count,
        "Ether": count,
        "Bomb": count,
        "Elixir": count
    }
}
```

Listing 3: Game State Structure

4 Game Archive Structure

The `game_archive` parameter provides a complete history of the battle, containing every turn played so far. This is extremely valuable for creating intelligent bots that can learn from past actions and adapt their strategies.

4.1 Archive Structure

Each entry in the `game_archive` list has the following structure:

```
{
    "player": {
        "hp": player_health_at_turn_start,
        "max_hp": player_max_health,
        "skill_points": player_sp_at_turn_start,
        "character_class": "Warrior"
    },
    "opponent": {
        "hp": opponent_health_at_turn_start,
        "max_hp": opponent_max_health,
        "skill_points": opponent_sp_at_turn_start,
        "character_class": "Mage"
    },
    "inventory": {
```

```
        "Potion": count_at_turn_start,
        "Ether": count_at_turn_start,
        "Bomb": count_at_turn_start,
        "Elixir": count_at_turn_start
    },
    "action": ["action_taken", "result_message"]
}
```

Listing 4: Game Archive Entry Structure

4.2 Using the Game Archive

4.3 Archive Analysis Strategies

- **Pattern Recognition**: Identify which skills/items your opponent uses most frequently
- **Health Trends**: Track how quickly each character's health is decreasing
- **Resource Management**: Monitor skill point usage and regeneration patterns
- **Timing Analysis**: Determine when opponents use healing items or defensive skills

5 Actions Reference

5.1 Basic Actions

- `"Attack"` - Perform a basic attack
- `"Defend"` - Defend against the next attack

5.2 Item Actions

- `"Use Potion"` - Heal 30 HP
- `"Use Ether"` - Heal 15 HP
- `"Use Bomb"` - Deal 40 damage
- `"Use Elixir"` - Fully restore HP

5.3 Class-Specific Skills

5.3.1 Warrior Skills

- `"Cast Power Strike"` - Heavy weapon attack (30-40 damage, 3 SP)
- `"Cast Shield Bash"` - Stuns and damages (20-30 damage, 2 SP)
- `"Cast Battle Cry"` - Heals HP (20-30 heal, 3 SP)
- `"Cast Whirlwind"` - Hits all enemies (25-35 damage, 4 SP)

5.3.2 Mage Skills

- `"Cast Fireball"` - High damage fire spell (35-45 damage, 5 SP)
- `"Cast Ice Shard"` - Freezing damage (25-35 damage, 3 SP)
- `"Cast Arcane Heal"` - Powerful healing (35-45 heal, 4 SP)
- `"Cast Lightning Bolt"` - Massive lightning damage (40-50 damage, 6 SP)

5.3.3 Rogue Skills

- `"Cast Backstab"` - High damage from behind (35-45 damage, 4 SP)
- `"Cast Poison Dart"` - Damage over time (0-10 damage + poison, 3 SP)
- `"Cast Shadow Dodge"` - Evade next attack (100 defense for 1 turn, 3 SP)
- `"Cast Quick Strike"` - Fast, low cost attack (25-35 damage, 2 SP)

6 Character Classes

6.1 Warrior

- **Max HP**: 240
- **Basic Attack**: 18-25 damage
- **Skill Points**: 4
- **Defense**: 15
- **Starting Inventory**: 4 Potions, 1 Ether, 1 Bomb, 1 Elixir

6.2 Mage

- **Max HP**: 160
- **Basic Attack**: 10-16 damage
- **Skill Points**: 8
- **Defense**: 25
- **Starting Inventory**: 2 Potions, 3 Ether, 2 Bombs, 1 Elixir

6.3 Rogue

- **Max HP**: 200
- **Basic Attack**: 15-22 damage
- **Skill Points**: 6
- **Defense**: 40
- **Starting Inventory**: 3 Potions, 2 Ether, 3 Bombs, 1 Elixir

7 Inventory Items

- **Potion**: Heals 35 HP
- **Ether**: Heals 20 HP
- **Bomb**: Deals 40 damage
- **Elixir**: Heals 80 HP (rare)

8 Strategy Guide

8.1 General Tips

1. **Monitor Health**: Use healing items when HP drops below 40
2. **Manage Skill Points**: Basic attacks restore SP for some classes
3. **Class Synergy**: Choose actions that complement your class strengths
4. **Opponent Awareness**: Consider the opponent's class and health when choosing actions
5. **Resource Management**: Don't waste powerful items or skills unnecessarily

8.2 Class-Specific Strategies

- **Warrior**: Use high-damage skills and defend to build skill points
- **Mage**: Manage skill points carefully and use powerful spells strategically
- **Rogue**: Use evasion and poison effects to control the battle

8.3 Skill Points

Skill points (SP) are utilized to cast skills which are unique to every class. The methods of regeneration of skill points for each class is as follows:

- **Warrior**: Using "Defend" during a turn regenerates 1 SP.
- **Mage**: Using "Attack" during a turn regenerates 2 SP.
- **Rogue**: Using "Attack" during a turn regenerates 1 SP.

8.4 Defense and Dodging

The stat "Defense" is different from the action "Defend". The action "Defend" decreases the amount of damage taken by the character upon receiving damage during the opponent's turn. The stat "Defense" refers to the probability of a character to dodge an attack, receiving no damage from it. The stat is unique to each class and goes as follows:

- **Warrior**: 15
- **Mage**: 25
- **Rogue**: 40

The probability of dodging an attack is $\frac{\text{Defense}}{100}$.

8.5 Status Effects

- **Poison**: Applied by Rogue's "Poison Dart". Deals 5 damage per stack at the start of each turn, then reduces stacks by 1.
- **Shadow Dodge**: Applied by Rogue's "Shadow Dodge" skill. Sets defense to 100 for 1 turn, guaranteeing dodges.

9 Advanced Bot Strategies

9.1 Using the Game Archive for Machine Learning

```python
def get_bot_action(current_game_state, game_archive):
    # Calculate damage per turn
    if len(game_archive) >= 2:
        current_hp = current_game_state['opponent']['hp']
        two_turns_ago = game_archive[-2]['opponent']['hp']
        damage_taken = two_turns_ago - current_hp
        # Adjust strategy based on damage output

    # Predict opponent's next move
    opponent_class = current_game_state['opponent']['character_class']
    opponent_skills = CLASS_SKILLS[opponent_class]

    # Check if opponent is low on skill points
    opponent_sp = current_game_state['opponent']['skill_points']
    likely_actions = []

    if opponent_sp < 3:
        likely_actions.extend(["Attack", "Defend"])
    else:
        # Add skills they can afford
        for skill, data in opponent_skills.items():
            if data.get("cost", 0) <= opponent_sp:
                likely_actions.append(f"Cast {skill}")

    # Counter the most likely actions
    if "Cast Fireball" in likely_actions and current_game_state['player
    ']['hp'] < 50:
        return "Defend"  # Defend against big attacks when low HP

    return "Attack"  # Default action
```

Listing 5: Advanced Archive Analysis

10 Example Bot

Here's the simple bot provided in the game:

```python
import random

def get_bot_action(current_game_state, game_archive):
    player_hp = current_game_state['player']['hp']
    inventory = current_game_state['inventory']

    # If HP is low and has a Potion, heal
    if player_hp < 40 and inventory.get("Potion", 0) > 0:
        return "Use Potion"

    # If opponent is low HP and we have a bomb, finish them!
```

```
    if current_game_state['opponent']['hp'] < 45 and inventory.get("Bomb", 0) > 0:
        return "Use Bomb"

    # Otherwise, cast a powerful spell or just attack
    if random.random() < 0.5: # 50% chance to cast Fireball
        return "Cast Fireball"
    else:
        return "Attack"
```

Listing 6: Example Bot Implementation

11 Testing Your Bot

1. Replace the `get_bot_action` function in `player1.py` or `player2.py` with your implementation
2. Run the game and select "AI vs AI" mode
3. Watch how your bot performs against the default opponent bot
4. Use the game archive analysis to debug and improve your strategy
5. Iterate and improve your bot based on performance!

12 Troubleshooting

- **Invalid Action**: Make sure your bot returns exactly the action strings shown in this manual
- **Skill Points**: Check that you have enough skill points before using skills
- **Inventory**: Verify item counts before trying to use items
- **Class Skills**: Remember that skills are class-specific
- **Game Archive**: Ensure you're accessing the archive correctly - it contains complete turn history
- **Performance**: If your bot is too slow, consider limiting how much archive history you analyze

Happy coding and may the best bot win!