Coding Club — Game Bot Manual

ChatGPT

October 21, 2025

1 Overview

Players must implement an AI (a "bot") that chooses one action per turn. The engine will call your bot with the current game state and a history of previous states; your bot must return a single action string each call. This manual documents the inputs, outputs, and the available game data (inventory and character presets) that your bot can use to decide actions.

2 Bot entry points

The engine expects two functions (present in `bots.py`):

```
def get_player_bot_action(current_game_state, game_archive):
    """
    Called for the player's bot.
    - current_game_state: dict describing current state (see section 4).
    - game_archive: list of previous game_state dicts (turn history).
    Must return a single action string (valid action strings listed in section 3).
    """

def get_opponent_bot_action(current_game_state, game_archive):
    """
    Called for the opponent bot.
    Same inputs/outputs as get_player_bot_action.
    """
```

Notes from the provided `bots.py`:

- The sample player's bot in the provided file uses a simple heuristic: if player's HP is below a threshold (40) and a Potion exists, it returns `"Use Potion"`. Otherwise it returns other actions.
- The sample opponent bot in the provided file returns `"Attack"` every turn.

3 Valid action strings

Bots must return **one string** per call. Valid strings include:

- `"Attack"`
- `"Defend"`
- `"Use <ItemName>"` e.g. `"Use Potion"` or `"Use Bomb"`

- `"Cast <SkillName>"` e.g. `"Cast Fireball"`
- (Menu/navigation strings sometimes used by menu-mode bots): `"NAV_SKILLS"`, `"NAV_ITEMS"`, `"NAV_MAIN"`

Important: Item and skill names must match exactly the names used by the engine (case and spacing). If your bot returns a `"Use ..."` for an item you don't have or `"Cast ..."` for a skill when you lack the required skill points, the engine will treat it as an invalid or wasted action.

4 Game state structure

When your function is called it receives two arguments. The essential fields in `current_game_state` to use are:

```
current_game_state = {
  "player": {
     "hp": int,
     "max_hp": int,
     "skill_points": int,
     "character_class": str,
  },
  "opponent": {
     "hp": int,
     "max_hp": int,
     "skill_points": int,
     "character_class": str,
  },
  "inventory": {
     "Potion": 3,
     "Ether": 1,
     "Bomb": 2,
     "Elixir": 0,
     ...
  }
}
```

The `game_archive` is a list of prior game states (turn history). Use it to detect opponent patterns or past actions (e.g., if opponent uses the same skill repeatedly).

5 Inventory presets

Below are the inventory definitions available to bots (from `inventory_presets.py`). These are the items, their counts and effects that a bot can inspect via `current_game_state["inventory"]`.

5.1 Base inventory

Item	Effect / Description	Base count
Potion	Heals 30 HP	3
Ether	Heals 15 HP	1
Bomb	Deals 40 damage	2
Elixir	Fully restores HP (full heal)	0

5.2 Class-specific starting inventories

(These are the starting inventory values that may be assigned for a selected class.)

Class	Item & effect	
Warrior	Potion (Heals 30 HP) Ether (Heals 15 HP) Bomb (Deals 40 damage) Elixir (Full restore)	
Mage	Potion (Heals 30 HP) Ether (Heals 15 HP) Bomb (Deals 40 damage) Elixir (Full restore)	(class-specific; typical presets give Mage more Ethers a
Rogue	Potion (Heals 30 HP) Ether (Heals 15 HP) Bomb (Deals 40 damage) Elixir (Full restore)	

Extra items (optional, may be present in some modes):

- **Mega Potion**: Heals 50 HP.
- **Smoke Bomb**: Utility/escape.
- **Antidote**: Cures poison.

Practical bot notes for items:

- Always check inventory counts before returning a `"Use <Item>"` action.
- `"Use Potion"` consumes one Potion and heals a fixed amount (30 HP).
- `"Use Bomb"` deals fixed direct damage to the opponent (40 HP).
- `"Use Elixir"` fully restores HP (consumes the Elixir).

6 Character presets and class skills

Below is the **exact** content you provided for `character_presets.py` and class skills. Include these in your bot logic when you inspect `current_game_state["player"]["character_class"]`.

6.1 Character presets (exact)

```
CHARACTER_PRESETS = [
    {
        "name": "Warrior",
        "max_hp": 120,
        "basic_attack_damage": (18, 25),
        "color": (200, 100, 50),
        "skill_points": 4,
        "defense": 15
    },
    {
        "name": "Mage",
```

```
        "max_hp": 80,
        "basic_attack_damage": (10, 16),
        "color": (100, 50, 200),
        "skill_points": 8,
        "defense": 25
    },
    {
        "name": "Rogue",
        "max_hp": 100,
        "basic_attack_damage": (15, 22),
        "color": (150, 150, 150),
        "skill_points": 6,
        "defense": 40
    }
]
```

6.2 Class-specific skills (exact)

```
# WARRIOR_SKILLS
WARRIOR_SKILLS = {
    "Power Strike": {"damage": (30, 40), "cost": 3, "description": "Heavy weapon attack"},
    "Shield Bash": {"damage": (20, 30), "cost": 2, "description": "Stuns and damages"},
    "Battle Cry": {"heal": (15, 25), "cost": 3, "description": "Heals and boosts morale"},
    "Whirlwind": {"damage": (25, 35), "cost": 4, "description": "Hits all enemies"}
}

# MAGE_SKILLS
MAGE_SKILLS = {
    "Fireball": {"damage": (35, 45), "cost": 5, "description": "High damage fire spell"},
    "Ice Shard": {"damage": (25, 35), "cost": 3, "description": "Freezing damage"},
    "Arcane Heal": {"heal": (30, 40), "cost": 4, "description": "Powerful healing"},
    "Lightning Bolt": {"damage": (40, 50), "cost": 6, "description": "Massive lightning damage"}
}

# ROGUE_SKILLS
ROGUE_SKILLS = {
    "Backstab": {"damage": (35, 45), "cost": 4, "description": "High damage from behind"},
    "Poison Dart": {"damage": (0, 10), "cost": 3, "description": "Damage over time"},
    "Shadow Dodge": {"defense": 100, "cost": 3, "description": "Evade next attack"},
    "Quick Strike": {"damage": (25, 35), "cost": 2, "description": "Fast, low cost attack"}
}

# CLASS_SKILLS mapping
CLASS_SKILLS = {
    "Warrior": WARRIOR_SKILLS,
```

```
    "Mage": MAGE_SKILLS,
    "Rogue": ROGUE_SKILLS
}
```

Each skill entry may include:

- `"damage"`: tuple (min,max) for damage done by the skill.
- `"heal"`: tuple (min,max) for heal amount.
- `"defense"`: numerical defense effect (e.g., 100 for Shadow Dodge).
- `"cost"`: skill point cost to cast the skill.
- `"description"`: text describing the skill.

7 Example bots

Below are two example bot implementations: (A) the simple heuristic from `bots.py` (as used in the provided file), and (B) a more robust template players can adapt.

7.1 (A) Simple heuristic (from `bots.py`)

```
import random

def get_player_bot_action(current_game_state, game_archive):
    """
    Simple logic:
    - If player HP < 40 and there is a Potion, use it.
    - Otherwise (fallback) choose attack or other actions.
    """
    player_hp = current_game_state['player']['hp']
    inventory = current_game_state['inventory']

    if player_hp < 40 and inventory.get("Potion", 0) > 0:
        return "Use Potion"

    # Fallback behaviour: attack each turn
    return "Attack"

def get_opponent_bot_action(current_game_state, game_archive):
    # Example opponent: always attack
    return "Attack"
```

7.2 (B) Improved playable template

This template demonstrates items and skills checks, and uses thresholds. Copy-paste and tune thresholds and skill names to match your strategy.

```
import random

def get_player_bot_action(current_game_state, game_archive):
    me = current_game_state["player"]
    opp = current_game_state["opponent"]
    inv = current_game_state["inventory"]
    cls = me["character_class"]
    sp = me["skill_points"]
```

```
9
10      # 1) Emergency heal
11      if me["hp"] <= max(30, int(me["max_hp"]*0.35)) and inv.get("Potion"
            , 0) > 0:
12          return "Use Potion"
13
14      # 2) Finish with Bomb if it's lethal
15      if inv.get("Bomb", 0) > 0 and opp["hp"] <= 40:
16          return "Use Bomb"
17
18      # 3) Class-specific skill usage (example for Mage)
19      if cls == "Mage" and sp >= 5:
20          # Cast Fireball if we have enough skill points
21          return "Cast Fireball"
22
23      # 4) If opponent is low, use basic Attack
24      if opp["hp"] <= 25:
25          return "Attack"
26
27      # 5) Otherwise random mix to avoid predictability
28      if random.random() < 0.7:
29          return "Attack"
30      else:
31          return "Defend"
```

8 Bot design heuristics and tips

- **Exact names:** Item and skill names returned by your bot must match the definitions in the presets.
- **Check resources:** Always check `inventory[item]` and `skill_points` before returning usage/cast actions.
- **Use the archive:** `game_archive` can reveal opponent patterns (e.g., always defends on even turns).
- **Poison / DOT:** `Poison Dart` is damage-over-time; plan heals accordingly.
- **Defensive skills:** `Shadow Dodge` (Rogue) sets defense very high for the next attack — use it to negate heavy incoming damage.
- **Determinism for testing:** Seed random in your bot for reproducible behavior during debugging (e.g., `random.seed(42)`).
- **Avoid wasted actions:** Returning a `"Use ..."` for a non-existent item or `"Cast ..."` when insufficient SP will typically waste your turn.

9 Quick reference (cheat sheet)

- **Function names:** `get_player_bot_action(current_game_state, game_archive)` and `get_opponent_bot_action(...)`
- **Action strings:** `Attack`, `Defend`, `Use <ItemName>`, `Cast <SkillName>`
- **Items:** Potion (30 HP), Ether (15 HP), Bomb (40 damage), Elixir (full heal), possible Mega Potion / Smoke Bomb / Antidote.

- **Classes:** Warrior, Mage, Rogue (see Character Presets section for HP, damage ranges, SP, defense).
- **Class skills:** See `CLASS_SKILLS` mapping in section 6.

10 Appendix: quick copy-paste of class skills

(Repeated for convenience; identical to section 6)

```
CLASS_SKILLS = {
    "Warrior": {
        "Power Strike": {"damage": (30, 40), "cost": 3, ...},
        "Shield Bash": {"damage": (20, 30), "cost": 2, ...},
        "Battle Cry": {"heal": (15, 25), "cost": 3, ...},
        "Whirlwind": {"damage": (25, 35), "cost": 4, ...}
    },
    "Mage": {
        "Fireball": {"damage": (35, 45), "cost": 5, ...},
        "Ice Shard": {"damage": (25, 35), "cost": 3, ...},
        "Arcane Heal": {"heal": (30, 40), "cost": 4, ...},
        "Lightning Bolt": {"damage": (40, 50), "cost": 6, ...}
    },
    "Rogue": {
        "Backstab": {"damage": (35, 45), "cost": 4, ...},
        "Poison Dart": {"damage": (0, 10), "cost": 3, ...},
        "Shadow Dodge": {"defense": 100, "cost": 3, ...},
        "Quick Strike": {"damage": (25, 35), "cost": 2, ...}
    }
}
```

Good luck and have fun coding your bots!