 Purpose  
Provide a concise, copy-and-paste-ready specification for a refactored Underwater Hockey simulation. This document will serve as a single source of truth you can refer back to as we rebuild the system from scratch.

 Goals

* **Modular**: Separate responsibilities into logical modules.
* **Maintainable**: Reduce global state, centralize configuration, avoid duplication.
* **Extensible**: Make it easy to add new behaviors (AI passes, defensive tactics, new UI elements).
* **Configurable**: Externalize “tuning” parameters (speeds, timers, formations) into JSON or a config module.
* **Clear Player Indexing**: Define a fixed mapping from unique IDs to teams and roles, independent of magic numbers.

 High-Level Architecture

arduino

CopyEdit

hockey\_sim/

├── config.py # constants, JSON loaders

├── data/

│ ├── green\_formations.json

│ └── blue\_formations.json

├── physics.py # collision & movement logic

├── player.py # Player class

├── ai.py # decision trees & AI actions

├── render.py # Tkinter setup & drawing routines

└── game.py # HockeyGame class with main loop

 Module Responsibilities

1. **config.py**
   * Define global constants: SCALE, SPRINT\_SPEED\_FACTOR, MAX\_DEPTH, FORMATION\_THRESHOLD, etc.
   * Provide load\_formations(path: str) → dict to read JSON and auto-generate wall variants.
2. **data/…json**
   * Each file maps formation names to offset dictionaries { "LF": [x,y], … }.
   * No hard-coded left/right variants—they’ll be mirrored at load time.
3. **physics.py**
   * project\_polygon, polygons\_collide (SAT)
   * is\_collision(player: Player, new\_x: float, new\_y: float) → bool
   * move\_toward(player: Player, tx: float, ty: float, threshold: float)
4. **player.py**
   * class Player:
     + Attributes: id: int, team: "green"|"blue", role: str, x,y,angle,depth,stamina
     + Methods: update\_position(dx,dy), update\_angle(θ), update\_color(), draw\_on(canvas)
5. **ai.py**
   * def decide\_action(player: Player, game\_state) → Action
   * Pluggable behaviors: score\_goal, pass\_to\_teammate, defend, etc.
6. **render.py**
   * def setup\_window(game: HockeyGame) to build Tk root, main canvas, status canvas
   * Drawing functions: pool, goals, benches, gauge bars
   * def render\_frame(game: HockeyGame) to update all canvas items
7. **game.py**
   * class HockeyGame:
     + **\_\_init\_\_**
       - Load config & formations
       - Initialize self.players, self.puck, self.score, timers, flags
       - Bind keys
     + **start()**
       - Call render.setup, then after(UPDATE\_INTERVAL, self.update)
       - root.mainloop()
     + **update()**
       - Handle input → update controlled player
       - Call physics.move\_all(self)
       - Update stamina/depth for every player
       - Resolve collisions & possession once
       - AI decisions & actions
       - render.render\_frame(self)
       - Schedule next after(UPDATE\_INTERVAL, self.update)

 Configuration & Tuning

* All numeric “magic” values live in config.py.
* Example entries:

python

CopyEdit

SCALE = 25

UPDATE\_INTERVAL = 50 # milliseconds

SPRINT\_SPEED\_FACTOR = 0.05

MAX\_DEPTH = 2.0 # meters

DEPTH\_STEP = 0.1 # m per frame

PASS\_FREEZE = 0.3 # seconds

COLLISION\_DEPTH\_THRESHOLD = 0.4

FORMATION\_THRESHOLD = 3 # pixels or meters?

* Formations in JSON; wall variants auto-generated.

 Player Indexing (fixed mapping)  
**Field Players**

| **Unique ID** | **Team** | **Role Label** | **Full Label** |
| --- | --- | --- | --- |
| 1 | green | FB | GreenFB |
| 2 | green | LB | GreenLB |
| 3 | green | RB | GreenRB |
| 4 | green | LF | GreenLF |
| 5 | green | C | GreenC |
| 6 | green | RF | GreenRF |
| 11 | blue | FB | BlueFB |
| 12 | blue | LB | BlueLB |
| 13 | blue | RB | BlueRB |
| 14 | blue | LF | BlueLF |
| 15 | blue | C | BlueC |
| 16 | blue | RF | BlueRF |

**Bench Players**

* *Blue bench* (top): IDs 21–24, labels “B” (first two), “F” (last two)
* *Green bench* (bottom): IDs 25–28, same labeling rule
* Assign team explicitly in code rather than inferring from ID.

 Next Steps

1. Create config.py + sample JSON for one formation.
2. Scaffold Player class in player.py and test instantiation/drawing.
3. Build HockeyGame skeleton in game.py with empty update().
4. Incrementally import physics.py, ai.py, and render.py, wiring each into update().
5. Verify that “Score a goal” AI and controlled-player movement work before adding passing, stamina, tackles.