規則策略函數說明書

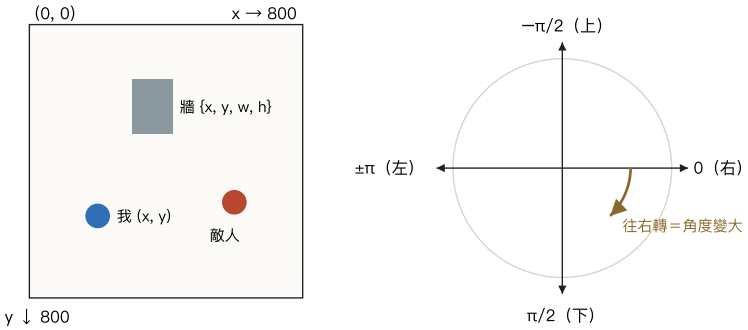
strategy_lib · 坦克競技場 · 神經網路與應用期中專題｜本文件由函數庫程式自動產生

這份說明書教你用 `strategy_lib` 寫規則策略 `strategy.py`。**第一次讀**：先看 A（觀念）和 B（常見任務），照著改範例就能動；要查某個函數的參數，再翻 D（函數說明卡）。說明書裡的每個範例都實際跑過。

A　先懂的觀念

A1　座標與角度

場地 800×800 像素，**原點在左上角**，x 往右變大、y **往下**變大。角度用**弧度**（π＝180°）：0 朝右、π/2 朝下、±π 朝左、−π/2 朝上；**往右轉＝角度變大**。程式裡通常不用自己算角度，用 `me.direction_to(目標)` 就好。



A2　坦克每一步要給 5 個數字

每 1/20 秒呼叫一次 `act(state)`，要回傳 5 個整數。通常不用自己寫數字，用 Action、Behavior 的函數加上 `combine()` 產生。

位置	部位（dict 的鍵）	0	1	2
第 1 個	移動 `move`	前進 `FWD`	後退 `BACK`	停止 `STOP`
第 2 個	車身 `body`	左轉 `LEFT`	不轉 `NONE`	右轉 `RIGHT`
第 3 個	砲塔 `turret`	左轉	不轉	右轉
第 4 個	開火 `fire`	不開火	開火	—
第 5 個	技能 `skill`	不按	按技能	—

例：`[0, 2, 1, 1, 0]`＝前進、車身右轉、砲塔不動、開火、不按技能。車身和砲塔各自轉，所以可以一邊往前開、一邊往旁邊開火。每一步車身最多轉約 4°、砲塔約 6.3°；前進速度每步 2.5～3.8 像素（依機體），倒車是前進的 0.6 倍。

A3　combine：把幾張「小紙條」合成 5 個數字

每個 Action、Behavior 函數只負責其中幾格，回傳一張只寫了那幾格的小紙條（dict）。`combine` 從「什麼都不做」的預設值開始，**照順序一張一張貼上去**，最後變成 5 個數字：

```
move  = Behavior.keep_distance(me, e, 230)   # 例如 {"move": 0, "body": 2}   只管移動、車身
aim   = Action.aim_turret_at(me, e)          # 例如 {"turret": 0}            只管砲塔
fire  = Action.fire_when_aimed(me, e)        # 例如 {"fire": 1}              只管開火
skill = Action.use_skill_if(me, False)       # 例如 {"skill": 0}             只管技能
return combine(move, aim, fire, skill)
```

步驟	移動	車身	砲塔	開火	技能
預設值（什麼都不做）	2 停	1 不轉	1 不轉	0	0
貼上 move	**0 前進**	**2 右轉**	1	0	0
貼上 aim	0	2	**0 左轉**	0	0
貼上 fire	0	2	0	**1 開火**	0
貼上 skill	0	2	0	1	**0**
結果	`[0, 2, 0, 1, 0]`＝前進、車身右轉、砲塔左轉、開火、不按技能				

沒給的格子不是 0，而是「什麼都不做」的預設值（combine 一定回傳 5 個數字）：

部位	移動	車身	砲塔	開火	技能
沒給的時候補	**2**（停；0 是前進）	**1**（不轉）	**1**（不轉）	**0**（不開火）	**0**（不按）

所以 `combine()` 什麼都不給＝`[2, 1, 1, 0, 0]`（原地不動）；傳進去的是 `None`（例如自訂行為這一步沒事做）會直接跳過。

同一格貼兩次，後面的蓋掉前面的：`combine(move, aim, fire, Action.hold_fire())` 裡，fire 先填「開火 1」，最後的 hold_fire 又填「開火 0」，所以一定不開火。想讓某個決定「最後說了算」，就放在最後面。

不用 combine 也可以：直接 `return [0, 2, 0, 1, 0]` 效果一樣。combine 只是讓你把「移動」交給 Behavior、「瞄準」交給 Action，各自決定好再合起來，不用記哪一格是第幾個。

A4　strategy.py 的骨架（每一行在做什麼）

```
from strategy_lib import Me, Memory, P, Action, Behavior, combine

# ① 常數與可調參數：寫在檔案最上面，名字用大寫，整場都不變
CHASSIS, MODULE = "standard", "boost"     # 你的配裝（網站下載的範本會自動填好）
KEEP_DIST = 230                           # 和敵人保持的距離（像素）
LOW_HP = 0.4                              # 血量低於多少就躲

class Strategy:                           # 類別名稱一定要叫 Strategy
    def reset(self):                      # ② 每局開始呼叫一次：在這裡建立「這局要記住」的變數
        self.memory = Memory()            #    Me 會用它記最後看到敵人的位置
        self.shots = 0                    #    自己的計數器：這局開了幾槍

    def act(self, state):                 # ③ 每一步（1/20 秒）呼叫一次，回傳 5 個整數
        me = Me(state, memory=self.memory)    # 把完整狀態整理成 me
        e = me.nearest_visible_enemy()        # 最近看得到的敵人，沒有就是 None

        # ④ 決定移動（這一步才用的變數，寫在 act() 裡就好）
        if me.outside_safe_zone(50):
            move = Behavior.go_to_safe_zone(me)
        elif e is None:
            move = Behavior.patrol(me)
        elif me.hp < LOW_HP:
            move = Behavior.hide_behind_wall(me, e)
        else:
            move = Behavior.keep_distance(me, e, KEEP_DIST)

        # ⑤ 瞄準、開火、技能
        aim = Action.aim_turret_at(me, e)
        fire = Action.fire_when_aimed(me, e)
        skill = Action.use_skill_if(me, me.bullet_coming())
        if fire["fire"]:
            self.shots += 1

        # ⑥ 合成 5 個整數回傳
        return combine(move, aim, fire, skill)
```

A5　變數要宣告在哪裡

要存什麼	寫在哪裡	例子
整場都不變的參數（距離、門檻）	檔案最上面，名字大寫	`KEEP_DIST = 230`
這局要一直記住的東西（下一局會清空）	`reset()` 裡寫 `self.名字 = 初始值`，或放在 `self.memory.名字`	`self.shots = 0`、`self.memory.prev_hp = 1.0`
只有這一步用的值	`act()` 裡直接寫	`e = me.nearest_visible_enemy()`

不要用檔案最上面的變數記「會變的東西」（例如上一步的血量）：它不會在每局開始時清空。`self.xxx` 一定要先在 `reset()` 給初始值，否則第一次在 `act()` 讀它會出錯（AttributeError）。

A6　資料長什麼樣子

東西	怎麼拿到	欄位（單位）
我	`me = Me(state, memory=self.memory)`	見 D1 的屬性表，例如 `me.x`、`me.hp`、`me.cooldown`
敵人（dict）	`me.nearest_visible_enemy()`、`me.visible`、`me.enemies`	詳見 A8 敵人資訊卡：`e["x"]`、`e["y"]`（像素）、`e["vx"]`、`e["vy"]`（每步移動的像素）、`e["hp_frac"]`（0～1）、`e["heading"]`、`e["turret"]`（弧度）、`e["stealthed"]`、`e["chassis"]`、`e["module"]`
子彈（dict）	`me.bullets`	詳見 A9 子彈資訊卡：`b["x"]`、`b["y"]`、`b["vx"]`、`b["vy"]`（子彈每步飛 11 像素，長砲 12.5）
牆（dict）	`me.walls`	詳見 A10 牆資訊卡：`w["x"]`、`w["y"]`（左上角）、`w["w"]`、`w["h"]`（寬、高）
位置	`P(x, y)`	`p.x`、`p.y`（注意：P 用 `.x`，dict 用 `["x"]`）

A7　清單怎麼用：visible、enemies、bullets、walls

清單（list）就是「很多個 dict 排在一起」：用 `for` 一個一個取出來，用 `[0]` 取第一個，用 `len()` 數有幾個。清單可能是空的（`[]`），用 `[0]` 之前先 `if 清單:` 檢查。 下面 A7～A11 的例子，都是從一場真的比賽（十字堡壘，第 29 步：看得到 2 台敵人、場上有 3 顆敵方子彈）抓下來的同一個瞬間，執行結果也是真的。

清單	裡面是什麼	由誰排序	這個瞬間
`me.visible`	敵人資訊卡（A8），只有看得到的	由近到遠	2 台
`me.enemies`	敵人資訊卡（A8），所有活著的，包含看不到、隱身中的	不排序	4 台
`me.bullets`	子彈資訊卡（A9），只有敵方的	不排序	3 顆
`me.walls`	牆資訊卡（A10），整局都不變	不排序	8 面

```
print(me.view)                          # 我的視線距離（雷達 450、其他 340）
print(len(me.visible), "台看得到、", len(me.enemies), "台還活著")

for e in me.visible:                    # 一台一台看：看得到的敵人，由近到遠
    print("敵人", round(e["x"]), round(e["y"]), "距離", round(me.distance_to(e)),
          "血", e["hp_frac"], e["chassis"], e["module"])

hidden = [e for e in me.enemies if e not in me.visible]     # 還活著但看不到的（牆後、太遠、隱身）
print("看不到的", len(hidden), "台")

if me.bullets:                          # 子彈清單可能是空的，先檢查
    b = min(me.bullets, key=me.distance_to)                 # 最近的一顆敵方子彈
    print("最近的子彈", round(me.distance_to(b)), "像素，速度", round(b["vx"], 1), round(b["vy"], 1))

w = me.walls[0]                         # 第一面牆：左上角 (x, y)、寬 w、高 h
print("牆", w["x"], w["y"], w["w"], w["h"], "共", len(me.walls), "面")
```

執行結果：

```
340
2 台看得到、 4 台還活著
敵人 420 133 距離 317 血 1 gunner boost
敵人 517 643 距離 338 血 1 scout stealth
看不到的 2 台
最近的子彈 406 像素，速度 -11.0 -0.7
牆 370 260 60 280 共 8 面
```

A8　敵人資訊卡（me.visible、me.enemies 裡的每一個）

每一台敵人是一個 dict，用 `e["欄位"]` 取值。`me.nearest_visible_enemy()` 回傳的也是同樣的資訊卡（或 None）。

欄位	型別	單位、範圍	意思	這個瞬間
`e["id"]`	int	1～4（10 台時 1～9）	這台敵人的編號，整局不變；可以用來追蹤同一台	1
`e["x"]`、`e["y"]`	float	像素，0～800	坦克中心的位置（左上角是原點，y 往下變大）	420.04、132.57
`e["vx"]`、`e["vy"]`	float	像素 / 步，約 −7.6～7.6	上一步 x、y 各移動了多少（停著或撞牆＝0）。速度大小＝`math.hypot(vx, vy)`；衝刺時是平常的 2 倍	−0.83、2.88
`e["heading"]`	float	弧度，−π～π	車身朝向（0 朝右、π/2 朝下），也是它前進的方向	1.85
`e["turret"]`	float	弧度，−π～π	砲口朝向，和車身各自轉；子彈會往這個方向射出	2.34
`e["hp_frac"]`	float	0～1	剩餘血量比例（1＝滿血）；每中一發扣 12 點，標準機體 100 血、重裝 120、輕裝 80、速射 90、長砲 100	1
`e["alive"]`	bool	—	還活著嗎；me.visible、me.enemies 只放活著的，所以一定是 True	True
`e["stealthed"]`	bool	—	正在隱身；隱身中的敵人不會出現在 me.visible（模型也看不到它）	False
`e["chassis"]`	str	standard、heavy、scout、gunner、lancer	機體：標準、重裝、輕裝、速射、長砲	gunner
`e["module"]`	str	stealth、shield、boost、antizone、radar	模組：隱身、護盾、衝刺、抗毒、雷達	boost

```
import math
from strategy_lib import wrap

e = me.visible[0]                                   # 最近的一台看得到的敵人
print("位置", round(e["x"]), round(e["y"]), "距離", round(me.distance_to(e)), "像素")
speed = math.hypot(e["vx"], e["vy"])                # 上一步移動了幾像素（停著或被擋住＝0）
print("速度", round(speed, 2), "像素/步")
side = wrap(me.direction_to(e) - me.heading)        # 相對我的車頭：正＝在右邊、負＝在左邊
print("在我的", "右邊" if side > 0 else "左邊", abs(round(math.degrees(side))), "度")
back = math.atan2(me.y - e["y"], me.x - e["x"])     # 從敵人看向我的方向
print("它的砲口離對準我還差", abs(round(math.degrees(wrap(back - e["turret"])))), "度")
print("10 步後大約在", round(e["x"] + e["vx"] * 10), round(e["y"] + e["vy"] * 10))
print("血", round(e["hp_frac"] * 100), "%｜配裝", e["chassis"], e["module"], "｜隱身中" if e["stealthed"] else "｜沒有隱身")
```

執行結果：

```
位置 420 133 距離 317 像素
速度 3.0 像素/步
在我的 右邊 42 度
它的砲口離對準我還差 94 度
10 步後大約在 412 161
血 100 %｜配裝 gunner boost ｜沒有隱身
```

> **注意**

> me.enemies 包含看不到的敵人，是規則策略的「特權」；神經網路比賽時只有觀測，看不到牆後的敵人。策略越依賴 me.enemies，模型越難學得像，一般建議用 me.visible 做決定。

A9　子彈資訊卡（me.bullets 裡的每一顆）

欄位	型別	單位、範圍	意思	這個瞬間
`b["x"]`、`b["y"]`	float	像素，0～800	子彈現在的位置	385.4、648.4
`b["vx"]`、`b["vy"]`	float	像素 / 步	每一步 x、y 各前進多少；方向＝`math.atan2(vy, vx)`，速度大小固定 11（長砲射的 12.5）	10.8、−1.9
`b["own"]`	bool	—	是不是我自己射的；me.bullets 只放敵方的，所以一定是 False	False

子彈的規則：從砲口前方 24 像素射出；每步飛 11 像素（長砲 12.5），最多飛 70 步（速射 55），所以射程約 770（速射 605、長砲 875）；碰到牆或場邊就消失；離某台坦克中心不到 16 像素就算打中，扣 12 點（護盾作用中會先擋掉）。打不到自己。

```
import math

for b in me.bullets:                                # 每一顆敵方子彈
    d = me.distance_to(b)
    speed = math.hypot(b["vx"], b["vy"])            # 11 像素/步（長砲 12.5）
    toward = (me.x − b["x"]) * b["vx"] + (me.y − b["y"]) * b["vy"] > 0    # 往我這邊飛嗎
    if toward:
        print("距離", round(d), "速度", round(speed, 1), "往我飛｜約", round(d / speed), "步後飛到我附近")
    else:
        print("距離", round(d), "速度", round(speed, 1), "遠離我｜不用理它")
print("會不會打到我（bullet_coming）：", me.bullet_coming())
```

執行結果：

```
距離 416 速度 11.0 往我飛｜約 38 步後飛到我附近
距離 406 速度 11.0 遠離我｜不用理它
距離 443 速度 11.0 往我飛｜約 40 步後飛到我附近
會不會打到我（bullet_coming）： False
```

A10　牆資訊卡（me.walls 裡的每一面）

欄位	型別	單位	意思	這個瞬間（第一面）
`w["x"]`、`w["y"]`	int	像素	牆的**左上角**	370、260
`w["w"]`、`w["h"]`	int	像素	寬（往右）、高（往下）。右下角＝(x＋w, y＋h)	60、280

牆是長方形，整局都不會變（十字堡壘 8 面、城區巷戰 9 面）。坦克開不過去、子彈打到就消失、視線會被擋住（me.can_see 就是在檢查有沒有牆擋）。

```
import math

w = me.walls[0]
print("左上角", (w["x"], w["y"]), "右下角", (w["x"] + w["w"], w["y"] + w["h"]), "中心", (w["x"] + w["w"] / 2, w["y"] + w["h"] / 2))

def inside(x, y, w):                                # 點 (x, y) 在不在這面牆裡
    return w["x"] <= x <= w["x"] + w["w"] and w["y"] <= y <= w["y"] + w["h"]
print("點 (400, 400) 在第一面牆裡嗎：", inside(400, 400, w))

def gap(w):                                         # 我到這面牆最近的距離（0＝貼著牆）
    dx = max(w["x"] − me.x, 0, me.x − (w["x"] + w["w"]))
    dy = max(w["y"] − me.y, 0, me.y − (w["y"] + w["h"]))
    return math.hypot(dx, dy)
near = min(me.walls, key=gap)
print("離我最近的牆", near, "距離", round(gap(near)), "像素")
```

執行結果：

```
左上角 (370, 260) 右下角 (430, 540) 中心 (400.0, 400.0)
點 (400, 400) 在第一面牆裡嗎： True
離我最近的牆 {'x': 670, 'y': 370, 'w': 40, 'h': 60} 距離 33 像素
```

A11　安全區（me.zone_center、me.zone_r）

屬性	型別	意思	這個瞬間
`me.zone_center`	P	安全區圓心，每局隨機，在地圖中心 (400, 400) 附近 ±80 像素內；整局不動	P(420, 320)
`me.zone_r`	float	安全區半徑（像素）。第 0～200 步是 620，之後越縮越小，第 1,000 步（50 秒）縮到 90 就不再縮	620

在圈外每秒扣 10 血（抗毒模組減半）。

```
c, r = me.zone_center, me.zone_r
edge = r − me.distance_to(c)                        # 正＝在圈內，離邊界還有多遠；負＝已經在圈外
print("圈心", round(c.x), round(c.y), "半徑", round(r), "｜我離邊界", round(edge), "像素，", "在圈內" if edge > 0 else "在圈外，每秒扣 10 血")
```

執行結果：

```
圈心 420 320 半徑 620 ｜我離邊界 378 像素， 在圈內
```

B　常見任務怎麼寫

下面的片段放在 `act()` 裡，假設前面已經有 `me`、`e = me.nearest_visible_enemy()`、`move`、`aim`、`fire`（見 A4 的骨架）。

B1　砲口轉向敵人、對準了就開火

最基本的攻擊：轉砲口和開火是兩件事，兩個都要寫。

```
aim = Action.aim_turret_at(me, e)          # 砲口轉向敵人（打提前量）
fire = Action.fire_when_aimed(me, e)       # 對準、裝填好、沒有牆擋，才開火
return combine(move, aim, fire)
```

B2　開到地圖上某個位置

目的地用 P(x, y)，x、y 是像素（0～800，左上角是原點）。

```
move = Action.drive_to(me, P(400, 400))              # 開到地圖中央
move = Action.drive_to(me, P(100, 700), arrive=30)    # 開到左下角附近，差 30 像素就算到
```

B3　往某個方向開（例如遠離敵人）

方向是角度（弧度）。敵人的方向加 π（180°）就是反方向。

```
import math
away = me.direction_to(e) + math.pi        # 敵人的反方向
move = Action.drive_in_direction(me, away)
```

B4　邊退邊打

倒車時車頭仍對著敵人，砲口另外瞄準。

```
move = Action.back_away_from(me, e)        # 車頭對著敵人往後退
return combine(move, Action.aim_turret_at(me, e), Action.fire_when_aimed(me, e))
```

B5　子彈飛來就開護盾（護盾模組）

護盾 2 秒內擋一發子彈，冷卻 20 秒，所以只在真的有子彈飛來時才開。

```
skill = Action.use_skill_if(me, me.bullet_coming())
return combine(move, aim, fire, skill)
```

B6　血少時隱身，隱身中不開火（隱身模組）

隱身中一開火就解除，所以隱身時要用 hold_fire 蓋掉開火。

```
skill = Action.use_skill_if(me, me.hp < 0.4 and e is not None)
if me.skill_active:
    fire = Action.hold_fire()                  # 隱身中不開火
return combine(move, aim, fire, skill)
```

B7　在圈外時衝刺回安全區（衝刺模組）

衝刺 1 秒內速度 ×2，期間不能開火，冷卻只有 6 秒。

```
if me.outside_safe_zone(0):
    move = Behavior.go_to_safe_zone(me)
    skill = Action.use_skill_if(me, True)      # 就緒就按
else:
    skill = Action.use_skill_if(me, False)
```

B8　記住上一步的血量，判斷是不是剛被打

每一步都會重新建立 me，要跨步記住的東西放在 self.memory。先在 reset() 裡給初始值：self.memory.prev_hp = 1.0

```
hurt = me.hp < self.memory.prev_hp           # 血比上一步少 → 剛被打
self.memory.prev_hp = me.hp                  # 記下來給下一步用
if hurt and e is not None:
    move = Behavior.hide_behind_wall(me, e)
```

B9　敵人躲起來時去找他

最近看過敵人但現在看不到，就開到最後看到的位置。

```
if e is None and me.saw_enemy_recently(80):
    move = Behavior.go_to_last_seen(me)
elif e is None:
    move = Behavior.patrol(me)
```

B10　只打血少的敵人

me.visible 是所有看得到的敵人（由近到遠），可以自己挑目標。

```
weak = [x for x in me.visible if x["hp_frac"] < 0.5]
target = weak[0] if weak else e              # 有血少的就打它，否則打最近的
aim = Action.aim_turret_at(me, target)
fire = Action.fire_when_aimed(me, target)
```

C　技能

模組	類型	效果	持續	冷卻	怎麼用
隱身塗層 `stealth`	主動	不出現在敵人的觀測裡；一開火就解除	2.5 秒	24 秒	危險時按，作用中用 `hold_fire()`
能量護盾 `shield`	主動	擋下一發子彈（12 點）	2 秒	20 秒	`me.bullet_coming()` 時按
推進衝刺 `boost`	主動	速度 ×2，期間不能開火	1 秒	6 秒	逃跑、回圈、拉近距離時按
抗毒塗層 `antizone`	被動	圈外傷害減半	一直有效		不用按
雷達 `radar`	被動	視線 340 → 450（別人也更早看到你）	一直有效		不用按

主動技能用 `Action.use_skill_if(me, 條件)`：條件成立而且 `me.skill_ready` 才會真的按。每局開始有 3 秒不能按。`me.skill_active` 表示技能正在作用中。範例見 B5～B7。

D　函數說明卡

D0　函數總覽（先看這張表找函數，再看下面的說明卡）

函數	做什麼
Me 的方法（me.…）（內建 13 個）	
`me.distance_to(p)`	我到 p 的直線距離。
`me.direction_to(p)`	從我指向 p 的方向。
`me.can_see(a, b)`	a 和 b 之間的直線有沒有被牆擋住。
`me.is_open(p, rad=18)`	p 這個位置能不能放下一台坦克（不撞牆、不出界）。
`me.clearance(ang)`	從我的中心往 ang 方向，多遠會碰到牆或場邊（像素）。和模型觀測的地形射線是同一種量法。
`me.wall_gap()`	我的車身邊緣離最近的牆或場邊還有多遠（像素）；0＝貼著牆。
`me.zone_route()`	這一局回安全區的路線（第一次呼叫時計算，存在 memory 裡）。Behavior.go_to_safe_zone 會用它繞過牆。
`me.nearest_visible_enemy()`	最近的一台看得到的敵人。
`me.in_safe_zone(p, margin=0)`	p 在不在安全區裡，而且離邊界至少 margin 像素。
`me.outside_safe_zone(margin=50)`	我是不是在安全區外，或離邊界不到 margin 像素（該往圈內走了）。
`me.saw_enemy_recently(n=80)`	最近 n 步內有沒有看到過敵人。
`me.bullet_coming(horizon=130, width=26)`	有沒有敵方子彈正朝我飛來。
`me.cover_spot(enemy, margin=28)`	找一個牆後面、敵人看不到、又在安全區內的位置。

Action：基本動作（內建 9 個）	
`Action.turn_body_toward(me, ang, tol=0.04)`	把車身轉向某個角度（只轉，不前進）。
`Action.drive_in_direction(me, ang)`	往某個方向開；前方被牆擋住會自動往兩側找出口。
`Action.drive_to(me, p, arrive=12)`	開到某個位置，到了就停。會自動轉向、繞過前方的牆。
`Action.back_away_from(me, p)`	車頭對著 p，往後倒車遠離它（倒車速度是前進的 0.6 倍）。適合邊退邊打。
`Action.stop()`	停下、車身不轉。
`Action.aim_turret_at(me, p, lead=True, tol=0.05)`	把砲口轉向目標。砲塔和車身分開轉，每一步最多轉約 6.3°。
`Action.fire_when_aimed(me, e, tol=0.12, check_walls=True)`	瞄準了才開火：同時滿足下面三個條件才回傳開火。
`Action.hold_fire()`	這一步不開火，例如隱身中（一開火隱身就解除）。放在 combine 的最後面就能蓋掉前面的開火。
`Action.use_skill_if(me, want)`	條件成立而且技能就緒時按技能。
Behavior：行為（內建 8 個）	
`Behavior.go_to_safe_zone(me, arrive=12)`	往安全區中心開（在圈外會持續扣血，每秒 10）。會繞過牆：圈心在牆裡時，開到圈心附近放得下坦克的位置。
`Behavior.keep_distance(me, e, d, near=40, far=60, approach=0.5)`	和敵人保持差不多 d 的距離，邊繞圈邊打。
`Behavior.rush_at(me, e, stop_at=62)`	衝向敵人，靠近到 stop_at 像素就停。
`Behavior.unstick(me, wait=1, back=16, gap=8, still=0.3, with_enemy=False, toward=None, pull=0.5, look=120, cooldown=0)`	脫困：貼著牆、幾乎沒在動時，往空曠的方向離開牆（牆在車頭就倒車，牆在車尾就前進）。
`Behavior.hide_behind_wall(me, e, margin=28)`	躲到牆後敵人看不到的地方；附近找不到牆就撤回安全區。
`Behavior.go_to_last_seen(me, arrive=12)`	開到最後一次看到敵人的位置（敵人躲起來時去找它）。
`Behavior.patrol(me, radius=300, spread=0.6)`	沒看到敵人時在安全區裡隨機巡邏，到了就換下一個點。
`Behavior.dodge(me)`	自己寫的第 2 層行為：找最近一顆朝自己飛來的子彈，往和子彈垂直的方向側移。
其他	
`combine(*parts)`	把各部位的決定合成坦克這一步的 5 個整數：[移動, 車身, 砲塔, 開火, 技能]。
`Me(state, memory=None)`	「我」這一步看得到、用得到的資訊。每一步在 act() 的第一行建立：me = Me(state, memory=self.memory)

`Memory()`	跨步保存的記憶。每一步都會重新建立 Me，只有存在 Memory（或 self）裡的東西會留到下一步。
`P(x, y)`	平面上的一個點，用來表示「要去的地方」。
`wrap(a)`	把角度換到 −π～π 之間（弧度）。
`@Action.custom`	把自己寫的基本動作加進 Action，之後用 Action.函數名(me, ...) 呼叫，和內建的一樣。
`@Behavior.custom`	把自己寫的行為加進 Behavior，之後用 Behavior.函數名(me, ...) 呼叫，和內建的一樣。
不是內建的：說明書和範例裡「自己寫」的函數（要先寫出來才能用）	
`Action.fire_if_close(me, e, max_dist=250)`	E 節的範例：用 @Action.custom 自己寫的，只在敵人夠近時開火
`Behavior.careful_fight(me, e, d=230)`	E 節的範例：用 @Behavior.custom 自己寫的，血多保持距離、血少躲牆後
`Behavior.dodge(me)`	範例 C 的 strategy.py 裡用 @Behavior.custom 自己寫的閃子彈
`want_skill(me, e, hurt)`	範例 C 的 strategy.py 裡自己寫的一般函數：判斷各模組什麼時候按技能

▼ D1　Me：我的資訊

`me = Me(state, memory=self.memory)`

「我」這一步看得到、用得到的資訊。每一步在 act() 的第一行建立：me = Me(state, memory=self.memory)

參數	預設	說明
`state`	（必填）	dict，act(state) 收到的完整狀態，原封不動傳進來
`memory`	None	Memory，在 reset() 建立的那一個（不給的話每步都是新的，就記不住東西）

屬性	說明
`t`	int，第幾步（每秒 20 步，一局最多 1,200 步）
`x、y`	float，我的位置（像素）
`heading`	float，車身朝向（弧度）
`vx、vy`	float，這一步實際移動了多少（像素）；speed：float，移動的距離（撞牆停住時是 0）
`turret`	float，砲口朝向（弧度，和車身各自轉）
`hp`	float，剩餘血量比例 0～1（0.5＝剩一半）
`cooldown`	int，還要幾步才能再開火，0＝可以開火
`skill_ready`	bool，技能可以按了（被動模組永遠 False）
`skill_active`	bool，技能正在作用中
`chassis、module`	str，我的配裝，例如 "heavy"、"shield"
`view`	int，我的視線距離（像素），雷達 450、其他 340
`visible`	list，看得到的敵人，由近到遠（每個都是敵人 dict）
`enemies`	list，所有活著的敵人，包含看不到、隱身中的
`bullets`	list，場上的敵方子彈（dict：x、y、vx、vy）
`walls`	list，牆，每面是 dict：x、y（左上角）、w（寬）、h（高）
`zone_center`	P，安全區圓心；zone_r：float，安全區目前的半徑（像素）
`mem`	Memory，就是你傳進來的 memory

範例

```
me = Me(state, memory=self.memory)
if me.hp < 0.3 and me.skill_ready: ...
```

`me.distance_to(p)`

我到 p 的直線距離。

參數	預設	說明
`p`	（必填）	要量的目標，可以是敵人 dict、子彈 dict 或 P

> **回傳**
> float：距離（像素）

範例

```
if e is not None and me.distance_to(e) < 200: ...   # 敵人在 200 像素內
```

`me.direction_to(p)`

從我指向 p 的方向。

參數	預設	說明
`p`	（必填）	目標，可以是敵人 dict、子彈 dict 或 P

> **回傳**
> float：絕對角度（弧度，−π～π；0 朝右、π/2 朝下）

範例

```
ang = me.direction_to(e)
move = Action.drive_in_direction(me, ang + math.pi)   # 往敵人的反方向開
```

`me.can_see(a, b)`

a 和 b 之間的直線有沒有被牆擋住。

參數	預設	說明
`a、b`	（必填）	兩個點，可以是 me、敵人 dict、子彈 dict 或 P

> **回傳**
> bool：True＝沒被擋住（看得到、子彈打得到）

範例

```
if me.can_see(me, e): ...                # 我和敵人之間沒有牆
```

`me.is_open(p, rad=18)`

p 這個位置能不能放下一台坦克（不撞牆、不出界）。

參數	預設	說明
`p`	（必填）	P，要檢查的位置
`rad`	18	float，坦克半徑加一點餘裕（像素），預設 18

> **回傳**
> bool：True＝放得下

`me.clearance(ang)`

從我的中心往 ang 方向，多遠會碰到牆或場邊（像素）。和模型觀測的地形射線是同一種量法。

參數	預設	說明
`ang`	（必填）	float，方向（弧度，絕對角度）；車頭正前方是 me.heading

> **回傳**
> float：距離（像素）

範例

```
if me.clearance(me.heading) < 30:      # 車頭前面 30 像素內就是牆
    ...
```

`me.wall_gap()`

我的車身邊緣離最近的牆或場邊還有多遠（像素）；0＝貼著牆。

> **回傳**
> float

範例

```
if me.wall_gap() < 6 and me.speed < 0.3: ...   # 貼著牆、幾乎沒在動
```

`me.zone_route()`

這一局回安全區的路線（第一次呼叫時計算，存在 memory 裡）。Behavior.go_to_safe_zone 會用它繞過牆。

`me.nearest_visible_enemy()`

最近的一台看得到的敵人。

「看得到」＝在我的視線距離內（340，雷達 450）、沒被牆擋、沒有隱身，和模型比賽時的觀測一致。

> **回傳**
> 敵人 dict 或 None（沒有看到任何敵人）
> 敵人 dict 的欄位：x、y（位置）、vx、vy（每步移動的像素）、heading、turret（弧度）、
> hp_frac（剩餘血量比例）、alive、stealthed、chassis、module

範例

```
e = me.nearest_visible_enemy()
if e is None:
    move = Behavior.patrol(me)
```

> **注意**
> 回傳可能是 None，用 e["x"] 之前要先檢查。Action 的函數收到 None 會自動「不轉、不開火」。

`me.in_safe_zone(p, margin=0)`

p 在不在安全區裡，而且離邊界至少 margin 像素。

參數	預設	說明
`p`	（必填）	P 或敵人 dict
`margin`	0	float，要離邊界多遠才算安全（像素），預設 0

> **回傳**
> bool

`me.outside_safe_zone(margin=50)`

我是不是在安全區外，或離邊界不到 margin 像素（該往圈內走了）。

參數	預設	說明
`margin`	50	float，離邊界多近就算危險（像素），預設 50；最多是安全區半徑的 30%（最小圈 90 時是 27）

> **回傳**
> bool：開局前 150 步（7.5 秒）一律回傳 False；圈心在牆裡、而我已經在圈心附近的空位時也回傳 False

範例

```
if me.outside_safe_zone(60):
    move = Behavior.go_to_safe_zone(me)
```

`me.saw_enemy_recently(n=80)`

最近 n 步內有沒有看到過敵人。

參數	預設	說明
`n`	80	int，往回算幾步，預設 80（4 秒）

> **回傳**
> bool：True 時 me.mem.last_seen 是最後看到的位置

範例

```
if e is None and me.saw_enemy_recently(60):
    move = Behavior.go_to_last_seen(me)
```

`me.bullet_coming(horizon=130, width=26)`

有沒有敵方子彈正朝我飛來。

參數	預設	說明
`horizon`	130	float，子彈再飛多遠以內才算（像素），預設 130（子彈每步飛 11，約 12 步內會到）
`width`	26	float，會從我旁邊多近擦過才算（像素），預設 26（坦克半徑 16）

> **回傳**
> bool

範例

```
skill = Action.use_skill_if(me, me.bullet_coming())   # 護盾：子彈飛來才開
```

`me.cover_spot(enemy, margin=28)`

找一個牆後面、敵人看不到、又在安全區內的位置。

參數	預設	說明
`enemy`	（必填）	敵人 dict，要躲開誰
`margin`	28	float，離牆多遠（像素），預設 28

> **回傳**
> P 或 None（附近沒有適合的牆）

▼ D2　Action：基本動作（第 1 層）

第 1 層：基本動作，每個函數只控制坦克的一兩個部位。

回傳的都是 dict，只寫要控制的部位：
move：0 前進、1 後退、2 停止
body：0 車身左轉、1 不轉、2 右轉
turret：0 砲塔左轉、1 不轉、2 右轉
fire：0 不開火、1 開火
skill：0 不按、1 按技能
最後用 combine() 把幾個 dict 合成 5 個整數。

`Action.turn_body_toward(me, ang, tol=0.04)`

把車身轉向某個角度（只轉，不前進）。

每一步車身最多轉約 4°（重裝 3.4°、輕裝 4.3°），所以轉半圈要 40 多步。

參數	預設	說明
`me`	（必填）	Me
`ang`	（必填）	float，目標角度（弧度，絕對角度，通常用 me.direction_to(某點) 算）
`tol`	0.04	float，差多少以內就不轉（弧度），預設 0.04（約 2°）

> **回傳**
> dict：{"body": 0 左轉 / 1 不轉 / 2 右轉}

`Action.drive_in_direction(me, ang)`

往某個方向開；前方被牆擋住會自動往兩側找出口。

方向和車頭差很多時，邊開邊轉（轉彎半徑約 43 像素）；車頭前面 45 像素內就是牆時，改成倒車轉向。
不會停下來原地轉：原地轉向時要往哪邊轉，取決於模型看不到的目的地，模型學了容易在原地打轉。

參數	預設	說明
`me`	（必填）	Me
`ang`	（必填）	float，要去的方向（弧度，絕對角度）

> **回傳**
> dict：{"move": 0 前進或 1 倒車, "body": 0 / 1 / 2}

範例

```
ang = me.direction_to(e) + math.pi          # 敵人的反方向
move = Action.drive_in_direction(me, ang)
```

`Action.drive_to(me, p, arrive=12)`

開到某個位置，到了就停。會自動轉向、繞過前方的牆。

目的地在 90 像素內又不在正前方時：在後方就倒車過去，否則停下來轉向（避免邊開邊轉繞過頭）。

參數	預設	說明
`me`	（必填）	Me
`p`	（必填）	P 或敵人 dict，目的地，例如 P(400, 400)
`arrive`	12	float，距離小於多少就算到了、停下（像素），預設 12

回傳

dict：{"move", "body"}

範例

```
move = Action.drive_to(me, P(400, 400))     # 開到地圖中央
```

注意

只管移動和車身，砲塔要另外用 aim_turret_at 控制。

`Action.back_away_from(me, p)`

車頭對著 p，往後倒車遠離它（倒車速度是前進的 0.6 倍）。適合邊退邊打。

參數	預設	說明
`me`	（必填）	Me
`p`	（必填）	要遠離的目標（敵人 dict 或 P）

回傳

dict：{"move": 1, "body": 0 / 1 / 2}

`Action.stop()`

停下、車身不轉。

回傳

dict：{"move": 2, "body": 1}

`Action.aim_turret_at(me, p, lead=True, tol=0.05)`

把砲口轉向目標。砲塔和車身分開轉，每一步最多轉約 6.3°。

參數	預設	說明
`me`	（必填）	Me
`p`	（必填）	要瞄準的目標：敵人 dict、P，或 None（None 時砲塔不動）
`lead`	True	bool，預設 True。是否打提前量：依子彈飛到目標的時間，瞄準敵人「待會會在」的位置； p 是 P（固定點）時沒有作用
`tol`	0.05	float，差多少以內就不轉（弧度），預設 0.05（約 3°）

回傳

dict：{"turret": 0 左轉 / 1 不轉 / 2 右轉}

範例

```
aim = Action.aim_turret_at(me, e)               # 瞄準最近的敵人
aim = Action.aim_turret_at(me, e, lead=False)   # 不打提前量
```

注意

只負責「轉」，開火要另外用 fire_when_aimed。

`Action.fire_when_aimed(me, e, tol=0.12, check_walls=True)`

瞄準了才開火：同時滿足下面三個條件才回傳開火。

```
1. 裝填完成（me.cooldown == 0；裝填 0.9 秒，速射 0.6、長砲 1.3）
2. 砲口和敵人（含提前量）的角度誤差小於 tol
3. 中間沒有牆擋住（check_walls=True 時）
```

參數	預設	說明
`me`	（必填）	Me
`e`	（必填）	敵人 dict 或 None（None 時不開火）
`tol`	0.12	float，角度誤差多少以內就開火（弧度），預設 0.12（約 7°）。越小越準、但開火越少
`check_walls`	True	bool，預設 True。False 時不管牆，對準就開火

回傳

dict：{"fire": 0 或 1}

範例

```
fire = Action.fire_when_aimed(me, e, tol=0.08)
```

注意

要先用 aim_turret_at 把砲口轉過去，否則一直對不準、不會開火。

`Action.hold_fire()`

這一步不開火，例如隱身中（一開火隱身就解除）。放在 combine 的最後面就能蓋掉前面的開火。

回傳

dict：{"fire": 0}

`Action.use_skill_if(me, want)`

條件成立而且技能就緒時按技能。

模組：隱身、護盾、衝刺是主動技能，要按；抗毒、雷達是被動能力，一直有效，按了也沒效果。
每局開始有 3 秒不能按；按下後要等冷卻（隱身 24 秒、護盾 20 秒、衝刺 6 秒）。

參數	預設	說明
`me`	（必填）	Me
`want`	（必填）	bool，你想不想用技能（寫成一個條件）

回傳

dict：{"skill": 0 或 1}（技能沒就緒時一定是 0，按了也是浪費）

範例

```
Action.use_skill_if(me, me.bullet_coming())                # 護盾：子彈飛來
Action.use_skill_if(me, me.hp < 0.4 and e is not None)     # 隱身：快死時躲
Action.use_skill_if(me, me.outside_safe_zone(0))           # 衝刺：在圈外時衝回去
```

`combine(parts)`

把各部位的決定合成坦克這一步的 5 個整數：[移動, 車身, 砲塔, 開火, 技能]。

依序合併，後面的會覆蓋前面同一個部位；沒有任何人指定的部位用預設值：停止、不轉、不轉、不開火、不按技能。

參數	預設	說明
`*parts`	（必填）	任意個 dict（或 None，會跳過）

回傳

list：5 個整數，act() 直接 return 它

範例

```
return combine(move, Action.aim_turret_at(me, e), Action.fire_when_aimed(me, e))
return combine(move, aim, fire, Action.hold_fire())   # 最後一個蓋掉開火 → 一定不開火
```

▼ D3　Behavior：行為（第 2 層）

第 2 層：行為，每個函數完成一件事（例如保持距離、躲牆後），會自己決定移動和車身。

回傳 dict（move、body），再和瞄準、開火、技能一起用 combine() 合起來。
參數都有預設值，不寫就用預設；想調整就用「參數名=值」，例如 Behavior.patrol(me, radius=200)。

`Behavior.go_to_safe_zone(me, arrive=12)`

往安全區中心開（在圈外會持續扣血，每秒 10）。會繞過牆：圈心在牆裡時，開到圈心附近放得下坦克的位置。

兩張地圖中央都是牆，約八成的局圈心在牆裡；直接 drive_to(me.zone_center) 會頂在牆邊、卡在十字的凹角。

參數	預設	說明
`me`	（必填）	Me
`arrive`	12	float，離目的地多近就停（像素），預設 12

> **回傳**
> dict：{"move", "body"}

`Behavior.keep_distance(me, e, d, near=40, far=60, approach=0.5)`

和敵人保持差不多 d 的距離，邊繞圈邊打。

```
距離 < d − near：倒車（車頭仍對著敵人）
距離 > d + far ：斜斜地靠近
中間：側移繞圈（不容易被打中）
```

參數	預設	說明
`me`	（必填）	Me
`e`	（必填）	敵人 dict（不能是 None）
`d`	（必填）	float，想保持的距離（像素），例如 230
`near`	40	float，預設 40
`far`	60	float，預設 60
`approach`	0.5	float，靠近時偏離正前方多少（弧度），預設 0.5（約 29°）

> **回傳**
> dict：{"move", "body"}

範例

```
move = Behavior.keep_distance(me, e, 230)
move = Behavior.keep_distance(me, e, 300, near=20, far=100)   # 長砲：拉遠一點
```

`Behavior.rush_at(me, e, stop_at=62)`

衝向敵人，靠近到 stop_at 像素就停。

參數	預設	說明
`me`	（必填）	Me
`e`	（必填）	敵人 dict
`stop_at`	62	float，停下的距離（像素），預設 62

> **回傳**
> dict：{"move", "body"}

`Behavior.unstick(me, wait=1, back=16, gap=8, still=0.3, with_enemy=False, toward=None, pull=0.5, look=120, cooldown=0)`

脫困：貼著牆、幾乎沒在動時，往空曠的方向離開牆（牆在車頭就倒車，牆在車尾就前進）。

為什麼需要：坦克停在牆邊的原因（等敵人、目的地在牆後、被擋住）常常是模型看不到的，
模型只學到「牆邊 → 停」，就會在牆邊左右轉個不停、整局不動。
觸發條件（牆很近、速度接近 0、看不看得到敵人）在 91 維觀測裡都看得到
（地形射線、自身速度、敵人欄位），錄進資料後模型學得到「卡在牆邊 → 離開」。

每一步都要呼叫（它要數「已經停了幾步」）；沒卡住時回傳 None，照原本的戰術走。
參數（都可以調，拿來做實驗）：
me：Me（要有 memory，才記得住停了幾步）
wait：int，貼牆又沒動幾步就算卡住，預設 1。等越久，資料裡「貼牆不動」標成「停」的越多，模型越學不會離開
back：int，離開時開幾步，預設 16（0.8 秒）。太短：剛離開又貼回去；太長：浪費時間
gap：float，車身離牆多近算「貼著牆」（像素），預設 8
still：float，這一步移動不到多少像素算「沒在動」，預設 0.3（全速約 2.5～3.8）
with_enemy：bool，看得到敵人時也觸發，預設 False（躲在牆後等敵人時不會被打斷）
toward：P、敵人 dict 或 None，離開時偏向哪裡，例如 me.zone_center（往圈內）、me.mem.last_seen（往最後看到敵人的地方）
pull：float，偏向 toward 的強度，0＝只看哪裡空曠，1＝很在意方向，預設 0.5
look：float，量「哪個方向空曠」時最多看多遠（像素），預設 120
cooldown：int，脫困結束後幾步內不再觸發，預設 0（避免在同一個角落反覆脫困可以設 20～40）

回傳

dict：{"move": 0 前進或 1 倒車, "body"}；沒卡住時 None

範例

```
unstick = Behavior.unstick(me)                                  # 放在戰術的最前面，每一步都呼叫
unstick = Behavior.unstick(me, toward=me.zone_center, pull=0.8)  # 脫困時盡量往圈內
if unstick:
    move = unstick
elif ...:
```

`Behavior.hide_behind_wall(me, e, margin=28)`

躲到牆後敵人看不到的地方；附近找不到牆就撤回安全區。

參數	預設	說明
`me`	（必填）	Me
`e`	（必填）	敵人 dict，要躲誰
`margin`	28	float，離牆多遠（像素），預設 28

回傳

dict：{"move", "body"}

範例

```
if me.hp < 0.4:
    move = Behavior.hide_behind_wall(me, e)
```

`Behavior.go_to_last_seen(me, arrive=12)`

開到最後一次看到敵人的位置（敵人躲起來時去找它）。

參數	預設	說明
`me`	（必填）	Me
`arrive`	12	float，預設 12

回傳

dict：{"move", "body"}

注意

先確認 me.saw_enemy_recently() 為 True，否則位置可能是 None。

`Behavior.patrol(me, radius=300, spread=0.6)`

沒看到敵人時在安全區裡隨機巡邏，到了就換下一個點。

參數	預設	說明
`me`	（必填）	Me
`radius`	300	float，巡邏點離安全區中心最遠多少（像素），預設 300
`spread`	0.6	float，最遠不超過安全區半徑的幾倍，預設 0.6

回傳

dict：{"move", "body"}

`Behavior.dodge(me)`

自己寫的第 2 層行為：找最近一顆朝自己飛來的子彈，往和子彈垂直的方向側移。

▼ D4　Memory、P、wrap

`self.memory = Memory()`

跨步保存的記憶。每一步都會重新建立 Me，只有存在 Memory（或 self）裡的東西會留到下一步。

在 reset() 裡建立新的，每局開始時就會清空：self.memory = Memory()

屬性	說明
`last_seen`	P 或 None，最後一次看到敵人的位置
`last_seen_t`	int，那是第幾步

`waypoint`	P 或 None，巡邏的目標點（Behavior.patrol 用）
`side`	+1 或 −1，這局偏好往右或往左繞（每局隨機） 也可以自己加屬性記東西（不用事先宣告）： self.memory.prev_hp = me.hp

範例

```
def reset(self):
    self.memory = Memory()
    self.memory.prev_hp = 1.0            # 自己加的：上一步的血量
```

`P(x, y)`

平面上的一個點，用來表示「要去的地方」。

參數	預設	說明
`x`	（必填）	float，像素，0（左）～800（右）
`y`	（必填）	float，像素，0（上）～800（下）

範例

```
center = P(400, 400)                     # 地圖正中央
move = Action.drive_to(me, center)
```

`wrap(a)`

把角度換到 −π～π 之間（弧度）。

兩個角度相減後常常超出範圍（例如 3.0 − (−3.0) = 6.0），用 wrap 換回來，
正值就是「往右轉比較近」，負值就是「往左轉比較近」，絕對值是差幾度（弧度）。

參數	預設	說明
`a`	（必填）	float，任意角度（弧度）

> **回傳**
> float：−π～π 之間的同一個角度

範例

```
diff = wrap(me.direction_to(e) - me.turret)   # 砲口還差多少才對準敵人
if diff > 0: print("要往右轉")
```

E　自己寫基本動作與行為

> **注意**
> 下面範例裡的 `fire_if_close` 、 `careful_fight` **不在函數庫裡**，是這段程式自己寫的。要先寫出 `@Behavior.custom` 和 `def careful_fight(...)` 那幾行，才能用 `Behavior.careful_fight(...)` ；只抄呼叫那一行會出現 AttributeError。範例 C 的 `dodge()` 也是這樣寫出來的。

在自己寫的函數上一行加 `@Action.custom` 或 `@Behavior.custom` ，之後就能和內建函數一樣用 `Action.名字(me, …)` 、 `Behavior.名字(me, …)` 呼叫。基本動作只控制想控制的部位；行為通常決定移動與車身，這一步沒事做就回傳 `None` 。名稱不能和內建的重複。

```
from strategy_lib import Me, Memory, Action, Behavior, combine

@Action.custom                                  # 自己寫的基本動作：加進 Action
def fire_if_close(me, e, max_dist=250):
    """只在敵人夠近、對準了才開火（遠距離子彈容易被閃掉）"""
    if e is None or me.distance_to(e) > max_dist:
        return {"fire": 0}
    return Action.fire_when_aimed(me, e)

@Behavior.custom                                # 自己寫的行為：加進 Behavior
def careful_fight(me, e, d=230):
    """血多就保持距離繞圈，血少就躲到牆後"""
    if me.hp < 0.4:
        return Behavior.hide_behind_wall(me, e)
    return Behavior.keep_distance(me, e, d, near=20, far=100)

class Strategy:
    def reset(self):
        self.memory = Memory()

    def act(self, state):
        me = Me(state, memory=self.memory)
        e = me.nearest_visible_enemy()
        if e is None:
            move = Behavior.patrol(me, radius=200)      # 內建行為也可以調參數
        else:
            move = Behavior.careful_fight(me, e, 250)   # 和內建行為一樣呼叫
        return combine(move, Action.aim_turret_at(me, e), Action.fire_if_close(me, e, 300))
```

E2　自己寫脫困規則（卡在牆邊時怎麼辦）

函數庫有內建的 `Behavior.unstick(me, …)`（見 D3），參數可以調。想用自己的想法，可以自己寫一個（名字要不一樣，例如 `my_unstick`）。寫法分四步：

1. **判斷「卡住」只用模型也看得到的線索**（第 2 段）
2. **連續卡幾步才算**（第 4 段）
3. **脫困要一次做完**：倒車並轉向比較空的一側，再往那個方向開一段（第 3、4 段）
4. **放在戰術最前面、每一步都呼叫**（第 5 段）

下面分 5 段寫出一個完整的 `strategy.py`，**由上往下接起來就是整個檔案**（《脫困練習》notebook 的 B 部分是同一份程式，可以直接執行）。例子故意用「直接朝圈心開、不避牆」的移動，圈心在牆後就會頂住。

第 1 段　檔頭和參數

`MY_UNSTICK`：脫困的 4 個參數放在最上面，方便實驗。`wait`＝連續卡幾步才算、`back`＝倒車幾步、`go`＝再往前開幾步、`near`＝車頭前面多近有牆算擋住（像素）。改成 `None` 就是不脫困。

```
import math
from strategy_lib import Me, Memory, Action, Behavior, combine

MY_UNSTICK = dict(wait=6, back=15, go=40, near=25)   # 脫困的參數；改成 None＝不脫困
```

第 2 段　① 判斷「卡住」

`is_stuck()` 回傳 True / False，三個條件都成立才算卡住。

- `me.clearance(me.heading)`：從車子中心往車頭方向，多遠會碰到牆或場邊。模型的觀測裡有 32 條地形射線，第 1 條就是車頭正前方，所以模型也看得到。
- `me.speed`：這一步實際移動了幾像素。全速是 2.5～3.8，撞牆停住時是 0。觀測裡有自身速度。
- `e is None`：看不到敵人。看得到敵人時多半是在交戰或躲牆，不該打斷。
- 不要用模型看不到的東西當條件（例如記憶裡的目標點、牆後敵人的位置），否則模型學不到「什麼時候該脫困」。

```
def is_stuck(me, e, near):
    """① 判斷這一步是不是「卡住」：只用模型也看得到的線索"""
    wall_ahead = me.clearance(me.heading) < near     # 車頭前面 near 像素內有牆（觀測：地形射線）
    not_moving = me.speed < 0.3                      # 這一步幾乎沒動（觀測：自身速度）
    no_enemy = e is None                             # 看不到敵人（觀測：敵人欄位）
    return wall_ahead and not_moving and no_enemy
```

第 3 段　往哪一側繞

`pick_side()` 回傳車身要怎麼轉（動作的第 2 個值：0＝左轉、1＝不轉、2＝右轉）。

- `me.heading - math.pi / 2`、`me.heading + math.pi / 2`：車頭往左、往右各轉 90° 的方向。畫面的 y 軸朝下，所以角度變大是往右轉。
- 比較左右兩邊多遠有牆，往比較空的那邊轉。
- 結果存在 `me.mem.esc_body`，**之後都用同一側**：每次重新挑的話，可能這次往左、下次往右，在牆前來回打轉。

```
def pick_side(me):
    """往哪一側繞：0＝左轉、2＝右轉。第一次卡住時挑比較空的一側，之後都用同一側（不會左右來回）"""
    if me.mem.esc_body is None:                      # 還沒決定過
        left = me.clearance(me.heading - math.pi / 2)    # 左手邊多遠有牆
        right = me.clearance(me.heading + math.pi / 2)   # 右手邊多遠有牆
        me.mem.esc_body = 0 if left > right else 2
    return me.mem.esc_body
```

第 4 段　② 數步數、③ 訂計畫並做完

`my_unstick()` 是脫困的本體，每一步都會被呼叫，依序檢查三件事：

- **正在脫困嗎？**（`esc_left > 0`）：照計畫做完。計畫分兩段，剩下的步數 ≥ `go` 時是第一段（倒車＋轉向），之後是第二段（往前直開）。回傳的 dict 只寫移動和車身：`{"move": 1, "body": …}` ＝倒車＋轉向、`{"move": 0, "body": 1}` ＝前進、車身不轉。
- **卡了幾步？**：`stuck_steps` 是「連續」卡住的步數，有一步沒卡就歸零。還不到 `wait` 步就回傳 `None`，交給後面的戰術決定。
- **卡住了**：先 `pick_side()` 決定方向，把 `esc_left` 設成 `back + go`（計畫總共幾步），這一步就開始倒車。
- 為什麼要第二段？只倒車的話，退開以後原本的移動又直直開回同一面牆（見下面的實測）。
- `@Behavior.custom`：把這個函數加進 `Behavior`，之後就能像內建行為一樣用 `Behavior.my_unstick(me, e, …)` 呼叫。

```
@Behavior.custom                                     # 加進 Behavior，之後用 Behavior.my_unstick(...) 呼叫
def my_unstick(me, e, wait=6, back=15, go=40, near=25):
    """自己寫的脫困：連續卡住 wait 步 → 倒車 back 步（同時轉向）→ 直直往前開 go 步"""
    m = me.mem
    # ③ 正在脫困：照計畫做完（esc_left＝計畫還剩幾步）
    if m.esc_left > 0:
        m.esc_left -= 1
        if m.esc_left >= go:
            return {"move": 1, "body": m.esc_body}   # 第一段：倒車（move 1）＋轉向
        return {"move": 0, "body": 1}                # 第二段：往前開（move 0）、車身不轉（body 1）
    # ② 數「連續」卡了幾步：有一步沒卡就歸零
    if is_stuck(me, e, near):
        m.stuck_steps += 1
    else:
        m.stuck_steps = 0
    if m.stuck_steps < wait:
        return None                                  # 還不算卡住：回傳 None，交給後面的戰術
    # 卡住了：訂計畫，這一步就開始倒車
    pick_side(me)
    m.esc_left = back + go
    m.stuck_steps = 0
    return {"move": 1, "body": m.esc_body}
```

第 5 段　④ 放進戰術

- `reset()`：每局開始呼叫一次。`my_unstick` 用到的 3 個記憶（`stuck_steps`、`esc_left`、`esc_body`）都要在這裡給初始值，否則第一步就會出現 AttributeError。
- `act()` 的第一件事就是呼叫 `my_unstick`：它要**每一步**都被呼叫，才數得到連續卡了幾步。回傳 `None` 表示沒卡住，交給後面的 if–elif。
- `**MY_UNSTICK`：把第 1 段的參數 dict 拆開傳進去，等於 `my_unstick(me, e, wait=6, back=15, go=40, near=25)`。
- 後面的移動故意不避牆：`turn_body_toward` 只轉車身，再把 `move` 設成 0（前進）。

```
class Strategy:
    def reset(self):                                 # 每局開始：自己加的記憶都要給初始值
        self.memory = Memory()
        self.memory.stuck_steps = 0                  # 連續卡了幾步
        self.memory.esc_left = 0                     # 脫困計畫還剩幾步（0＝沒在脫困）
        self.memory.esc_body = None                  # 往哪一側繞（第一次卡住時決定）

    def act(self, state):
        me = Me(state, memory=self.memory)
        e = me.nearest_visible_enemy()
        # ④ 脫困放在戰術最前面，每一步都呼叫（它要數步數）
        unstick = Behavior.my_unstick(me, e, **MY_UNSTICK) if MY_UNSTICK is not None else None
        if unstick:
            move = unstick
        elif e is None:
            # 自己寫的移動：直接朝圈心開、不避牆（圈心在牆後就會頂住）
            move = Action.turn_body_toward(me, me.direction_to(me.zone_center))
            move["move"] = 0
        else:
            move = Behavior.keep_distance(me, e, 230)
        return combine(move, Action.aim_turret_at(me, e), Action.fire_when_aimed(me, e))
```

實測

用離線小場地測（一面牆擋在我和圈心中間、沒有假想坦克，走 15 秒）：

MY_UNSTICK	移動距離	最後離圈心	看到什麼

`None`（不脫困）	24	136	頂在牆上不動
`go=0`（只倒車）	446	158	退開又開回同一面牆，一直重複
預設（`go=40`）	853	**32**	繞過牆，到達圈心附近
`go=20`	400	150	繞得不夠遠，又撞回同一面牆
`wait=40`	751	73	等比較久才脫困，比較慢到

動手改：`go` 改 20、`near` 改 15、`wait` 改 40，什麼時候繞不過去？把 `pick_side` 改成每次重新挑，會不會在牆前來回？拿掉 `no_enemy`，看得到敵人時也脫困好不好？想放進自己的策略：把第 1～4 段的函數和 `MY_UNSTICK` 貼到 `strategy.py` 上面，`reset()` 加 3 個記憶，`act()` 最前面加呼叫。

注意：函數庫的 `drive_to`、`drive_in_direction`、`go_to_safe_zone` 本來就會避牆、繞牆，用它們移動時很少卡住；自己寫的脫困最常派上用場的是**自己寫的移動**，或是**故意貼著牆等待**的戰術。規則策略不卡住，不代表模型不卡住：開與不開各錄一次、各訓練一次，看天梯轉播比較。

F　常見錯誤

現象	原因與改法
`TypeError: 'NoneType' object is not subscriptable`	沒看到敵人時 `e` 是 None，卻用了 `e["x"]`。先寫 `if e is not None:`。
`AttributeError: 'Strategy' object has no attribute …`	`self.xxx` 沒在 `reset()` 給初始值。
坦克一直原地打轉	目的地在牆裡或圈外到不了，或每一步給的方向一直變。先用 `me.is_open(P(…))` 檢查目的地。
規則策略正常，**訓練出來的模型**卻常在牆邊停住、原地打轉	模型只看得到 91 維觀測。規則策略在牆邊停下的原因（等敵人、目的地在牆後）模型看不到，它只學到「牆邊 → 停」。改法：少用 `Action.stop()`，移動交給 `drive_to`、`drive_in_direction`（邊開邊轉，不會原地停下來轉）；也可以在戰術最前面加 `Behavior.unstick(me)`（貼牆不動就馬上離開牆），再比較開與不開訓練出來的模型。
從來不開火	只寫了 `fire_when_aimed`、沒寫 `aim_turret_at`（砲口沒轉過去），或 `tol` 設太小。
開火打在牆上	用了 `check_walls=False`；預設會檢查牆。
技能從來沒用	被動模組（抗毒、雷達）本來就不用按；主動模組檢查條件是否太嚴，開局 3 秒內也按不出來。
combine 後動作不是我要的	後面的會覆蓋前面同一個部位，把「最後要算數」的放在最後。