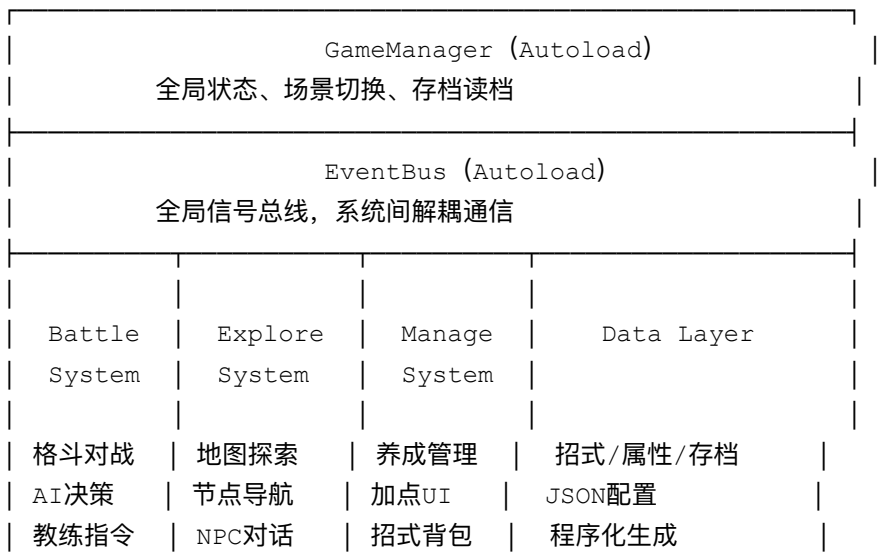
Stancelore - 技术开发概述

1. 引擎与环境

项目	选型	说明
引擎	Godot 4.4+	开源，原生支持 Windows/macOS/Linux 导出
主语言	GDScript	与引擎深度集成，适合快速原型
编辑器	Godot 内置 + VSCode（GDScript 插件）	双屏开发：Godot 改场景，VSCode 写逻辑
像素美术	Aseprite	sprite sheet 导出，支持动画标签
版本控制	Git + GitHub	.gitignore 排除 .godot/ 缓存目录
Steam 集成	GodotSteam 插件	成就、云存档、Overlay
分辨率	384×216 基础，4x 缩放到 1536×864	像素完美渲染，接近 16:9

---

2. 架构总览



```
| 打击判定   | 遭遇触发    | 升级结算   |  数据持久化          |
|           |            |           |                     |
└──────────┴────────────┴──────────┴─────────────────────┘
```

核心原则

- **数据驱动**：招式、敌人、关卡、对话全部 JSON 配置，不硬编码
- **信号解耦**：系统间通过 EventBus 通信，不互相持有引用
- **场景独立**：每个系统是独立场景，可单独运行调试
- **Resource 优先**：用 Godot 的 Resource 系统做数据结构，原生支持序列化

---

3. Autoload 全局单例

项目依赖 3 个 Autoload：

GameManager

职责：
- 持有当前游戏状态（当前区域、玩家数据引用）
- 场景切换（`battle → result → map`）
- 存档/读档入口
- 游戏流程状态机：`TITLE → EXPLORING → BATTLE → RESULT → EXPLORING`

EventBus

职责：全局信号中转

信号清单（`MVP` 阶段）：

```
- battle_started(enemy_data)
- battle_ended(result: BattleResult)
- move_learned(move: MoveData)
- level_up(new_level: int)
- coach_directive_changed(directive: String)
- node_entered(node_data)
- stat_points_changed()
```

PlayerData

职责：

- 持有玩家角色的所有持久数据
- 属性点分配
- 招式背包和装备槽
- 当前等级/经验
- 已探索的节点记录
- 提供存档序列化/反序列化

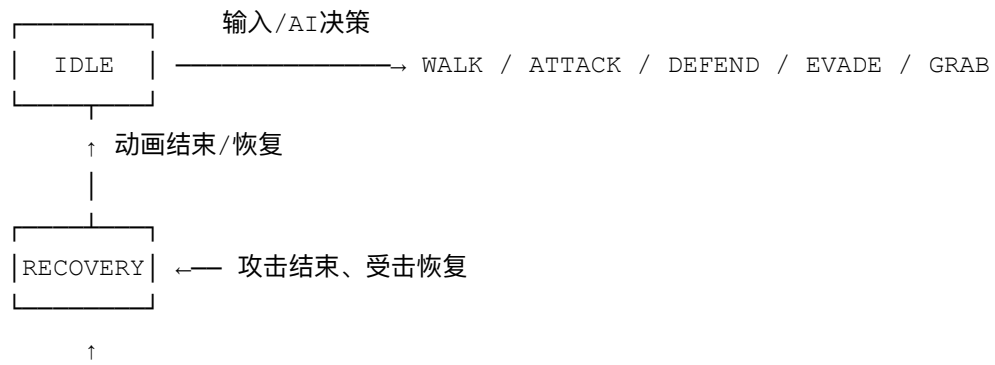
---

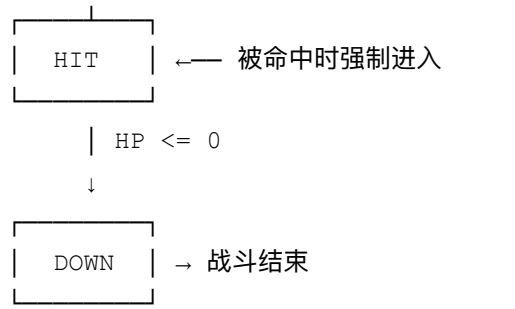
4. 战斗系统（核心模块）

4.1 场景结构

```
BattleScene (Node2D)
├── Background (Sprite2D)
├── FighterLeft (CharacterBody2D)          # 玩家角色
│   ├── Sprite (AnimatedSprite2D)
│   ├── HurtBox (Area2D → CollisionShape2D)    # 被打判定
│   ├── StateMachine (Node)
│   └── AIBrain (Node)
├── FighterRight (CharacterBody2D)         # 敌人
│   ├── （同上结构）
├── HitEffects (Node2D)                    # 特效池
├── BattleHUD (CanvasLayer)
│   ├── HPBarLeft / HPBarRight
│   ├── CoachPanel                         # 教练指令按钮
│   ├── Timer Display
│   └── MoveLearnNotify                    # 领悟提示
└── BattleManager (Node)                   # 战斗流程控制
    ├── RoundTimer
    └── ResultCalculator
```

4.2 角色状态机





状态转换规则：
- IDLE：可以转到任何行为状态
- ATTACK：不可中断（除非被命中）→ 打完进 RECOVERY
- DEFEND：可随时取消回 IDLE
- HIT：强制状态，受击硬直结束后回 IDLE
- RECOVERY：收招硬直，不可行动，结束后回 IDLE

4.3 AI 决策流程

每个决策周期（约 15-30 帧触发一次，避免每帧决策导致行为抖动）：

1. 感知（Perception）
   ├── 计算双方距离 → CLOSE / MID / FAR
   ├── 读取自身 HP 比例 → HEALTHY / HURT / DANGER
   ├── 读取对手当前状态 → IDLE / ATTACKING / DEFENDING / STUNNED
   └── 读取自身能量值

2. 查表（Decision Table Lookup）
   └── table[distance][hp_state][opponent_state] → 基础概率分布
       例：table[CLOSE][HEALTHY][IDLE] = {ATK:40, DEF:25, EVD:15, CTR:10, GRB:10}

3. 属性修正（Stat Modifier）
   ├── STR → ATK 概率 +，GRB 概率 +
   ├── AGI → EVD 概率 +，CTR 概率 +
   ├── VIT → DEF 概率 +，ATK 概率 -
   ├── INT → SPECIAL 权重 +
   ├── DEX → 连招后续概率 +
   └── LUK → 所有低概率行为 +

4. 教练指令修正（Coach Modifier）
   ├── 猛攻 → ATK × 1.8, DEF × 0.3
   ├── 压制 → ATK × 1.3, DEF × 0.7
   ├── 均衡 → 无修正
   ├── 防守 → DEF × 1.8, ATK × 0.4
   ├── 回避 → EVD × 2.0, ATK × 0.3
   └── 反击 → CTR × 2.5, DEF × 1.3, ATK × 0.3

5. 归一化 → 总和缩放到 100%

6. 加权随机选择行为类别

7. 从已装备招式中，筛选该类别 + 当前距离可用的招式，随机选一个执行

4.4 碰撞判定

两套碰撞体：

```
HurtBox (Area2D) — 角色身体，被打区域
  └── collision_layer = 2 (HURTBOX)

HitBox (Area2D) — 攻击时临时激活的攻击判定
  └── collision_mask = 2 (检测 HURTBOX)
  └── 仅在攻击动画的 active frames 期间启用
  └── 命中后立即禁用，防止多段判定（除非招式设计为多段）
```

碰撞层规划：

```
Layer 1: 物理碰撞（地面、墙壁）
Layer 2: HurtBox
Layer 3: HitBox
Layer 4: 弹射物（远程招式）
```

命中处理流程：

```
HitBox 进入 HurtBox →
读取攻击方招式数据（伤害、击退、硬直）→
施加伤害 →
触发 hit stop（Engine.time_scale = 0 持续 3-5 帧）→
屏幕震动（Camera2D offset 随机偏移）→
播放受击动画 + 击退 →
判定招式领悟概率
```

4.5 打击感实现

```
# 三件套：hit_stop + screen_shake + 击中特效

# Hit Stop（时间暂停）
func apply_hit_stop(duration_frames: int = 4):
    Engine.time_scale = 0.0
    await get_tree().create_timer(duration_frames / 60.0, true, false, true).time
    Engine.time_scale = 1.0

# Screen Shake
```

```
func apply_screen_shake(intensity: float = 3.0, duration: float = 0.15):
    var tween = create_tween()
    for i in range(6):
        var offset = Vector2(randf_range(-intensity, intensity), randf_range(-int
        tween.tween_property(camera, "offset", offset, duration / 6.0)
    tween.tween_property(camera, "offset", Vector2.ZERO, 0.05)

# 命中时统一调用
func on_hit(attacker: Fighter, defender: Fighter, move: MoveData):
    apply_hit_stop(move.hit_stop_frames)
    apply_screen_shake(move.damage * 0.5)
    spawn_hit_effect(defender.global_position)
    defender.take_damage(move)
```

---

5. 数据层

5.1 数据结构关系

```
PlayerData (Resource)
├── name: String
├── level: int
├── exp: int
├── stats: StatBlock
│   ├── str: int
│   ├── agi: int
│   ├── vit: int
│   ├── int_: int
│   ├── dex: int
│   └── luk: int
├── stat_points_available: int
├── equipped_moves: Array[MoveData]     # 8 个槽
├── move_inventory: Array[MoveData]     # 背包
└── explored_nodes: Array[String]       # 已访问节点 ID

MoveData (Resource)
├── id: String                          # 唯一标识
├── template_id: String                 # 基础模板引用
├── type: enum (ATK/DEF/EVD/GRB/SPE)
├── range: enum (CLOSE/MID/FAR)
├── base_damage: float
├── speed_frames: int                   # 出招帧数
├── recovery_frames: int                # 收招帧数
├── hit_stun_frames: int
```

```
├── knockback: Vector2
├── affixes: Array[AffixData]
├── rarity: int
└── animation_key: String

AffixData (Resource)
├── id: String
├── display_name: String                # "灼热的"
├── effect_type: enum (DAMAGE_MULT/LIFESTEAL/ARMOR_BREAK/FREEZE/...)
├── value: float                        # 效果数值
└── element: enum (NONE/FIRE/ICE/POISON/LIGHTNING/DARK/WIND)

EnemyData (Resource)
├── id: String
├── display_name: String
├── stats: StatBlock
├── equipped_moves: Array[String]       # 招式 ID 列表
├── ai_profile: String                  # 决策表 profile 名
├── sprite_sheet: SpriteFrames
├── loot_table: Array[LootEntry]        # 掉落表
└── description: String

StageNode (Resource)
├── id: String
├── type: enum (BATTLE/TOWN/BOSS/DISCOVERY/HIDDEN)
├── display_name: String
├── description: String
├── connections: Array[String]
├── unlock_condition: Dictionary
├── enemy_id: String                    # BATTLE/BOSS 类型
├── npcs: Array[String]                 # TOWN 类型
└── rewards: Array[String]              # DISCOVERY 类型
```

5.2 JSON 配置文件结构

```
data/
├── moves/
│   ├── templates.json       # 基础模板定义（20-30个）
│   │   [{ "id": "straight_punch", "type": "ATK", "range": "CLOSE",
│   │      "base_damage": 10, "speed_frames": 8, "recovery_frames": 12,
│   │      "hit_stun_frames": 10, "knockback": [80, -20],
│   │      "animation_key": "attack_punch" }, ...]
│   │
│   ├── affixes.json         # 词缀池
│   │   [{ "id": "burning", "display_name": "灼热的",
│   │      "effect_type": "DAMAGE_MULT", "value": 1.3,
```

```
│   │      "element": "FIRE", "rarity_weight": 50 }, ...]
│   │
│   └── rare_moves.json      # 手工设计的稀有招式
│       [{ "id": "time_stop_counter", "display_name": "时停反击",
│          "type": "SPE", "range": "CLOSE", ... }, ...]
│
├── enemies/
│   ├── forest.json          # 魔法森林敌人
│   └── ruins.json           # 王国废墟敌人
│
├── stages/
│   ├── world_map.json       # 世界地图区域列表
│   ├── starter_village.json
│   ├── magic_forest.json
│   └── kingdom_ruins.json
│
├── dialogues/
│   ├── old_hunter.json
│   └── ruins_scholar.json
│
└── ai_profiles/
    ├── default.json         # 默认 AI 概率表
    ├── aggressive.json      # 攻击型 profile
    ├── defensive.json       # 防御型 profile
    └── boss_forest.json     # Boss 专用 profile
```

5.3 存档系统

存档路径：`user://save/slot_X.json`

存档内容：

```
{
  "player": { PlayerData 序列化 },
  "explored_nodes": ["node_id_1", "node_id_2", ...],
  "current_location": "magic_forest:hunters_lodge",
  "play_time_seconds": 3600,
  "version": "0.1.0"
}
```

策略：

- 手动存档（据点城镇可存档）
- 对战结束自动存档
- JSON 格式方便调试，后期可加密

---

6. 探索系统

6.1 节点地图渲染

```
NodeMap (Control)
├── MapBackground (TextureRect)
├── NodeContainer (Control)
│   ├── MapNode × N (TextureButton)      # 每个节点
│   │   ├── Icon (TextureRect)           # 类型图标
│   │   ├── Label (Label)                # 节点名
│   │   └── LockOverlay (ColorRect)      # 未解锁遮罩
├── ConnectionLines (Line2D × N)         # 节点间连线
├── PlayerMarker (Sprite2D)              # 当前位置指示
└── DescriptionPanel (PanelContainer)    # 节点详情面板
```

逻辑：
- 从 JSON 读取节点数据
- 节点位置在 JSON 中预定义（x, y 坐标）
- 已访问节点正常显示，未访问相邻节点半透明，远处节点隐藏
- 点击相邻节点 → 移动 PlayerMarker → 触发节点事件
- BATTLE 节点 → 切换到 BattleScene
- TOWN 节点 → 弹出 NPC 对话面板
- HIDDEN 节点 → 检查解锁条件

6.2 场景切换流程

```
MapScene ──[选择战斗节点]──→ 转场动画 ──→ BattleScene
                                              │
                                          对战结束
                                              │
                                              ↓
MapScene ←──[回到地图]─── 转场动画 ←── ResultScreen
                                          (经验/升级/招式领悟)
```

使用 SceneTree.change_scene_to_packed() 切换
转场用简单的 ColorRect fade in/out

---

7. 招式程序化生成

```
# move_generator.gd 核心逻辑
```

```
func generate_move(rarity: int, element_bias: String = "") -> MoveData:
    # 1. 随机选基础模板
    var template = pick_random_template()

    # 2. 根据品质决定词缀数量
    var affix_count = rarity + 1  # 白0→1个, 蓝1→2个, 紫2→3个

    # 3. 从词缀池随机选取（考虑元素偏向和不重复）
    var affixes = pick_affixes(affix_count, element_bias)

    # 4. 创建 MoveData 实例
    var move = MoveData.new()
    move.id = generate_uid()
    move.template_id = template.id
    move.type = template.type
    move.range = template.range
    move.animation_key = template.animation_key
    move.rarity = rarity

    # 5. 应用词缀修正数值
    move.base_damage = template.base_damage
    move.speed_frames = template.speed_frames
    move.recovery_frames = template.recovery_frames
    move.hit_stun_frames = template.hit_stun_frames
    move.knockback = template.knockback

    for affix in affixes:
        move.affixes.append(affix)
        apply_affix_effect(move, affix)

    # 6. 生成显示名："灼热的迅捷直拳"
    move.display_name = build_display_name(template, affixes)

    return move
```

---

8. 关键 Godot 节点和功能使用

需求	Godot 方案
角色移动/物理	CharacterBody2D + move_and_slide()
角色动画	AnimatedSprite2D（sprite sheet 导入自 Aseprite）
碰撞判定	Area2D（HitBox/HurtBox）+ collision layer/mask

状态机	自写轻量状态机（Node + Dictionary），不用插件
AI 决策定时	Timer 节点，0.3-0.5 秒触发一次决策
打击暂停	Engine.time_scale 控制
屏幕震动	Camera2D offset + Tween
粒子特效	GPUParticles2D（击中火花、元素特效）
UI 布局	Control 节点树 + Theme 统一样式
节点地图	Control + TextureButton + Line2D
对话系统	RichTextLabel + 自写对话状态机
数据持久化	FileAccess JSON 读写到 user://
场景转场	AnimationPlayer（fade）+ SceneTree.change_scene
音效	AudioStreamPlayer2D（战斗音效带空间感）
BGM	AudioStreamPlayer（全局，Autoload 管理）

---

9. 编码规范

文件命名：`snake_case.gd / snake_case.tscn`
类命名：`PascalCase`
变量/函数：`snake_case`
常量：`UPPER_SNAKE_CASE`
信号：`past_tense_verb (battle_ended, move_learned)`

目录规则：
- `scenes/` 放 `.tscn` 文件
- `scripts/` 放 `.gd` 文件（与 `scene` 分离，便于复用）
- `data/` 放 `JSON` 配置
- `assets/` 放美术和音频资源

注释规则：
- 每个类顶部写 `class_name` 和一句话说明
- 公开函数写简短 `doc comment`
- 复杂逻辑写行内注释
- `TODO/FIXME` 标记待完善的地方

10. Phase 0 启动检查清单

- ☐ 安装 Godot 4.4 stable
- ☐ 创建项目 stancelore，设置分辨率 384×216，stretch mode = viewport
- ☐ 创建文件夹结构：scenes/ scripts/ data/ assets/
- ☐ 注册 3 个 Autoload：GameManager，EventBus，PlayerData
- ☐ 搭建 BattleScene：
  - ☐ 两个 ColorRect placeholder 角色
  - ☐ CharacterBody2D + CollisionShape2D
  - ☐ Area2D HitBox / HurtBox
  - ☐ 简单地面（StaticBody2D）
- ☐ 实现 fighter_state_machine.gd（IDLE/WALK/ATTACK/DEFEND/EVADE/HIT/RECOVERY）
- ☐ 实现 decision_table.gd（硬编码初始概率表）
- ☐ 实现 ai_brain.gd（感知→查表→修正→随机→执行）
- ☐ 实现 coach_system.gd（3 按钮 + 5 秒 Timer）
- ☐ 实现 hit_stop + screen_shake
- ☐ 添加简易 HP 条（ProgressBar）
- ☐ 反复测试调参，直到对战好看