# Project Documentation – *Prince of Persia EDU*

Auto‑generated 2025‑06‑16

## 0. Table of contents

1. [Project structure](#Xb38fe0144eaf114d6469ca01a284990847254f1)
2. [Settings constants](#X9b0f80f92d24233ff0519ce5b3afdd5862784b4)
3. [Utilities](#X39bffed68eff8545334bc7d727250ac082fe0bf)
4. [Game loop & state machine](#X22d7800c679e1a8b4b5345bbf0424044644ebd6)
5. [Sprite system](#X2e28c7da28f6c612dad2f0e5127524437a22b17)
6. [Level loader & camera](#X8485350a437001829865434947f1de5672c6144)
7. [UI layer](#X12714eb6fe197d808f9b2c206110a430b94df8d)
8. [Data files](#X447c85d11347f2a1ee8a386fd7d5f35dcbe751c)
9. [Level‑file format](#X2069fdae83725298fed9dc232c38f2dab833247)

## 1. Project structure

src/  
 ├─ core/  
 │ ├─ game.py # main loop, StateMachine  
 │ ├─ settings.py # global constants  
 │ └─ state\_machine.py # tiny FSM helper  
 │  
 ├─ entities/ # every gameplay sprite  
 │ ├─ base.py # BaseEntity(animation+timer)  
 │ ├─ player.py # Player  
 │ ├─ guard.py # stationary enemy (example)  
 │ ├─ bandit.py # moving melee AI  
 │ ├─ skeleton.py # harder enemy AI  
 │ ├─ bat.py # flying enemy  
 │ ├─ ladder.py # exit object  
 │ └─ trap.py # SpikeTrap, FloorCollapse, BladeSpinner  
 │  
 ├─ levels/  
 │ └─ level.py # Level class (loader, camera, scoring)  
 │  
 ├─ ui/  
 │ ├─ menu.py # pygame‑menu based splash  
 │ └─ hud.py # on‑screen HUD (hp, time, score)  
 │  
 ├─ utils/  
 │ ├─ loader.py # image/sound cache, audio helpers  
 │ ├─ save.py # JSON high‑score handling  
 │ └─ \_\_init\_\_.py  
└─ assets/ # all external data  
 ├─ images/ … # sprites, tiles, backgrounds  
 ├─ sounds/ … # .mp3 & .wav SFX  
 └─ levels/level01.txt # ascii maps

## 2. core.settings

| Constant | Type | Purpose |
| --- | --- | --- |
| WIDTH, HEIGHT | int | native window size (964×570) |
| TILE | int | base tile height (128) |
| FPS | int | target framerate (60) |
| DEFAULT\_VOLUME | float | 0.6 – initial master volume |
| ENEMY\_POINTS | int | 100 – score per enemy kill |
| TIME\_BONUS | dict[int,int] | {0:0, 30:500, 60:250, 90:100} |
| MUSIC\_FILE | Path | *assets/sounds/music.mp3* |
| SND\_JUMP, SND\_PUNCH, SND\_GAME\_OVER | Path | SFX paths |
| IMG\_DIR, SND\_DIR, LVL\_DIR, CFG\_DIR, SAVE\_DIR | Path | resource roots |

## 3. Utilities (utils)

### loader.py

* `→Surface` – cached image loader.
* `→Sound` – cached SFX loader.
* `` – one‑shot playback.
* `` – propagate master volume change to mixer.
* `` – break sprite‑sheets into frames.
* , – helpers.

### save.py

* **File**: *save/scores.json*
* `– returns{level: {nick:score}}` dict.
* `` – append + keep top‑5 per level.

## 4. Core loop

### core.game.Game

| Field | Description |
| --- | --- |
| screen | main display surface |
| clock | framerate timer |
| states | StateMachine controlling Menu / Level |
| level\_index | current level id |
| levels | list[str] level file names |
| nick | player nickname ( editable in Options ) |
| player | reference injected by Level |

* `– instantiatesLevel` and switches FSM.
* `` – switches to menu & plays SFX.

### core.state\_machine.StateMachine

Minimal wrapper with .state, .change(new) delegating to sub‑state’s handle\_event/update/draw.

## 5. Entity classes

Below: only public attributes & overriden methods.

### BaseEntity(Sprite)

| Field | Note |
| --- | --- |
| animations: dict[state, list[Surface]] | frame sets |
| state: str | current key in animations |
| anim\_interval: float | seconds per frame |
| frame\_t: float | timer |

### Player(BaseEntity)

| Field | Purpose |
| --- | --- |
| pos: Vector2 | sub‑pixel world position |
| vel: Vector2 | current planar velocity |
| facing: Vector2 | unit dir for attack hitbox |
| hp, max\_hp | health pool |
| score | accumulated score (kills + bonus) |
| jumping, falling\_off | vertical states |
| … | … |

| Method | Summary |
| --- | --- |
| handle\_input() | keyboard mapping, state changes |
| attack\_hitbox() | returns Rect for sword swing 0–0.15 s |
| update(dt) | physics + animation + invulnerability blink |

### Bandit / Skeleton (enemy AI)

Shared behaviour:

* patrol random direction (\_pick\_new\_dir) until player inside `` then chase.
* Moves in iso via `; each tentative step validated byLevel.\_foot\_ok` mask test.
* update(dt) handles turn timer, move, melee check (calls Level.\_apply\_damage).

Unique constants: SPD\_PATROL, DMG, MAX\_HP …

### Trap classes

* SpikeTrap (toggle up/down) – damage\_enabled True when spikes raised
* FloorCollapse – falls after first contact (trigger)
* BladeSpinner – continuous rotation, constant damage field

## 6. Level class – main responsibilities

| Section | Key members |
| --- | --- |
| **Loading** | parses ASCII text, spawns sprites, builds floor\_surface & floor\_mask |
| **Camera** | camera: Vector2, \_edge\_scroll\_camera() keeps player within ¼ edge |
| **Physics** | loop detects if sprite stands on opaque pixel of floor\_mask |
| **Scoring** | score tally, time bonus on ladder, enemy kill detection |
| **Audio** | calls play(SND\_GAME\_OVER) on death |

## 7. UI layer

### ui.menu.MainMenu

Buttons: **NEW GAME**, **SCORES**, **OPTIONS**, **QUIT**.  
*Scores* sub‑menu shows top‑5 of each level on blue background (non‑scrollable).

### ui.hud.HUD

Displays:

* HP: current/max (blinks red on hit)
* Level: n
* Time: ⧖elapsed s
* Score: value

## 8. Data & Assets

assets/  
 ├─ images/  
 │ ├─ iso\_block/iso\_plate\_[12].png # floor tiles  
 │ ├─ iso\_prince\_hero\*.png # player spritesheets  
 │ ├─ iso\_bandit\_\*.png # bandit  
 │ ├─ iso\_skeleton\_contrast\_\*.png # skeleton  
 │ ├─ traps/\*.png # spikes, blade, collapse  
 │ └─ bgn/bgn.png # background  
 ├─ sounds/  
 │ ├─ music.mp3 jump.mp3 punch.mp3 game\_over.mp3  
 └─ levels/  
 level01.txt level02.txt level03.txt

## 9. Level‑file ASCII format

Each character corresponds to one isometric **plate tile** (size 128 × 64 px in projection). Unknown chars are ignored.

| Symbol | Spawns / Action |
| --- | --- |
| P | Player start |
| # | Empty (void) |
| G | Guard enemy |
| B | Bandit enemy |
| S | Skeleton enemy |
| b | Bat enemy |
| L | Ladder (level exit) |
| ^ | SpikeTrap |
| X | FloorCollapse |
| O | BladeSpinner |

Tiles are read top‑to‑bottom; screen‑space is computed by:

wx = (x - y) \* TILE/3 + WIDTH/2 + offset\_x  
wy = (x + y) \* TILE/6 + 100 + offset\_y

offset\_\* recentre large maps.

A 3‑wide path can be authored like:

###  
###  
###

Ladders should be reachable; void is any area filled with #.

*End of documentation*