Awesome—let's scaffold your Godot 4.4.1 project so you can drop in scenes and immediately run, with a clean path to expand. I'm giving you minimal, runnable scripts (player movement + 4-way attacks with a real hitbox, NPCs that auto-register and flip hostile when attacked, interactables for doors/chests/items), plus the exact Project Settings and a quick "first run" checklist. (All mapped to your notes.)

---

1) Project settings (once)

Project > Project Settings > Input Map
Create these actions and bind keys:

- `move_up` (W, Up)
- `move_down` (S, Down)
- `move_left` (A, Left)
- `move_right` (D, Right)
- `interact` (E)
- `attack` (Space)
- `toggle_killer_mode` (K)
- `pause_menu` (Esc)

Project > Project Settings > AutoLoad

- Add `GameManager.gd` (path: `res://scripts/GameManager.gd`) as a singleton named **GameManager**.

(Optional but helpful)

- **Display > Window > Size**: Width 1280, Height 720.
- **Layer names** (Project Settings > Layer Names > 2D):
    - 1: `world`
    - 2: `player`
    - 3: `npc`

- 4: `hitbox`
- 5: `interactable`

---

2) File/folder layout

```
res://
  scenes/
    Player.tscn
    NPC.tscn
    Door.tscn
    Chest.tscn
    Level_1.tscn
  scripts/
    GameManager.gd
    BaseEntity.gd
    Player.gd
    NPC.gd
    InteractableObject.gd
```

---

3) Scripts

GameManager.gd (autoload)

```
# res://scripts/GameManager.gd
extends Node

var players: Array = []
var npcs: Array = []
var interactables: Array = []

var paused := false

func _ready() -> void:
	# Report a moment after the scene finishes loading
	call_deferred("_report_loaded_counts")

func register_entity(entity: Node, kind: String) -> void:
	match kind:
		"player":
			if not players.has(entity):
				players.append(entity)
		"npc":
			if not npcs.has(entity):
				npcs.append(entity)
```

```
                "interactable":
                        if not interactables.has(entity):
                                interactables.append(entity)

func unregister_entity(entity: Node) -> void:
        players.erase(entity)
        npcs.erase(entity)
        interactables.erase(entity)

func set_paused(state: bool) -> void:
        paused = state
        get_tree().paused = state

func _report_loaded_counts() -> void:
        print("Loaded -> Players:", players.size(),
              " | NPCs:", npcs.size(),
              " | Interactables:", interactables.size())
```

BaseEntity.gd

```
# res://scripts/BaseEntity.gd
extends CharacterBody2D
class_name BaseEntity

@export var hp: int = 10
@export var max_hp: int = 10
@export var defense: int = 0
@export var speed: float = 120.0
@export var silver_pieces: int = 0
@export var inventory_slots: int = 6
@export var inventory: Array[String] = []

# Name/type metadata hooks
@export var npc_type: String = ""
@export var ai_type: String = "" # "idle","wander","patrol" (for NPCs)
@export var has_dialogue: bool = false
@export var dialogue_text: String = ""

var is_dead := false

func take_damage(amount: int, from: Node) -> void:
        if is_dead: return
        var dmg := max(0, amount - defense)
        hp -= dmg
        if hp <= 0:
                _die()

func _die() -> void:
        is_dead = true
        # TODO: play "death" animation if present
        queue_free()
```

Player.gd

```
# res://scripts/Player.gd
extends BaseEntity

@onready var anim: AnimatedSprite2D = $AnimatedSprite2D
@onready var interaction_area: Area2D = $InteractionArea
var killer_mode := false
var last_dir := Vector2.DOWN

# Attack hitbox spawned at runtime (simple & minimal)
const ATTACK_LIFETIME := 0.12
const ATTACK_SIZE := Vector2(16, 16)
const ATTACK_DAMAGE := 3

func _ready() -> void:
	set_collision_layer_value(2, true) # player
	set_collision_mask_value(1, true)  # world
	set_collision_mask_value(3, true)  # npc
	GameManager.register_entity(self, "player")
	# Default idle
	if anim and anim.has_animation("idle"):
		anim.play("idle")

func _unhandled_input(event: InputEvent) -> void:
	if event.is_action_pressed("toggle_killer_mode"):
		killer_mode = !killer_mode
		print("Killer Mode:", killer_mode)
	elif event.is_action_pressed("pause_menu"):
		GameManager.set_paused(!GameManager.paused)

func _physics_process(delta: float) -> void:
	if GameManager.paused:
		velocity = Vector2.ZERO
		return

	var input_dir :=
Input.get_vector("move_left","move_right","move_up","move_down")
	velocity = input_dir * speed

	if input_dir != Vector2.ZERO:
		last_dir = input_dir.normalized()
		_play_walk_anim(last_dir)
	else:
		_play_idle_anim()

	if Input.is_action_just_pressed("attack"):
		_try_attack(last_dir)

	move_and_slide()
```

```
func _play_walk_anim(dir: Vector2) -> void:
	if not anim: return
	# 4-way walk animations: walk_up, walk_down, walk_right (flip_h for left)
	if abs(dir.x) > abs(dir.y):
		if dir.x > 0:
			anim.flip_h = false
			if anim.has_animation("walk_right"):
anim.play("walk_right")
		else:
			anim.flip_h = true
			if anim.has_animation("walk_right"):
anim.play("walk_right")
	else:
		if dir.y < 0 and anim.has_animation("walk_up"):
			anim.flip_h = false
			anim.play("walk_up")
		elif anim.has_animation("walk_down"):
			anim.flip_h = false
			anim.play("walk_down")

func _play_idle_anim() -> void:
	if anim and anim.has_animation("idle"):
		anim.play("idle")

func _try_attack(dir: Vector2) -> void:
	# Requires a direction (Space+direction rule); if no direction—do nothing
	if dir == Vector2.ZERO:
		return

	# Choose proper attack animation if present
	if abs(dir.x) > abs(dir.y):
		if dir.x > 0 and anim.has_animation("attack_right"):
			anim.flip_h = false; anim.play("attack_right")
		elif dir.x < 0 and anim.has_animation("attack_right"):
			anim.flip_h = true; anim.play("attack_right")
	else:
		if dir.y < 0 and anim.has_animation("attack_up"):
			anim.flip_h = false; anim.play("attack_up")
		elif anim.has_animation("attack_down"):
			anim.flip_h = false; anim.play("attack_down")

	_spawn_attack_hitbox(dir)

func _spawn_attack_hitbox(dir: Vector2) -> void:
	var hb := Area2D.new()
	hb.name = "AttackHitbox"
	hb.collision_layer = 0
	hb.collision_mask = 0
	hb.set_collision_layer_value(4, true) # hitbox layer
	hb.set_collision_mask_value(3, true)  # collides with NPCs
	add_child(hb)

	var shape := CollisionShape2D.new()
	var rect := RectangleShape2D.new()
```

```
        rect.size = ATTACK_SIZE
        shape.shape = rect
        hb.add_child(shape)

        # Position hitbox just ahead of the player based on dir
        var offset := dir.normalized() * (ATTACK_SIZE.x + 8.0)
        hb.position = offset

        hb.body_entered.connect(func(body: Node):
                if body is BaseEntity and body != self:
                        # Respect friendly rules: only allow if NPC is hostile or
we're in killer_mode
                        var allow := true
                        if body.has_meta("is_friendly") and
bool(body.get_meta("is_friendly")):
                                allow = killer_mode
                        if allow:
                                (body as BaseEntity).take_damage(ATTACK_DAMAGE,
self)
        )

        # Clean up shortly
        await get_tree().create_timer(ATTACK_LIFETIME).timeout
        if is_instance_valid(hb):
                hb.queue_free()

func interact_nearest() -> void:
        # Simple proximity interact (E)
        for area in interaction_area.get_overlapping_areas():
                if area.get_parent() and
area.get_parent().has_method("on_interact"):
                        area.get_parent().on_interact(self)
                        return
```

Note

- Attacks require a direction (matches your “Space + direction” idea).
- You can’t attack friendly NPCs unless `killer_mode` is ON; hostile NPCs ignore that restriction.

NPC.gd

```
# res://scripts/NPC.gd
extends BaseEntity

@onready var anim: AnimatedSprite2D = $AnimatedSprite2D
@onready var interaction_area: Area2D = $InteractionArea
```

```
@export var is_hostile: bool = false
@export var wander_range: float = 64.0
@export var patrol_points: Array[Vector2] = []
@export var is_friendly: bool = true # start friendly; turns hostile if hurt

var _home_pos := Vector2.ZERO
var _wander_target := Vector2.ZERO

func _ready() -> void:
	set_meta("is_friendly", is_friendly)
	set_collision_layer_value(3, true) # npc
	set_collision_mask_value(1, true)  # world
	_home_pos = global_position
	GameManager.register_entity(self, "npc")
	if anim and anim.has_animation("idle"):
		anim.play("idle")
	_pick_new_wander()

func _physics_process(delta: float) -> void:
	if GameManager.paused or is_hostile:
		velocity = Vector2.ZERO
		move_and_slide()
		return

	# Minimal "wander" AI if selected
	if ai_type == "wander" or ai_type == "":
		var to_target := (_wander_target - global_position)
		if to_target.length() < 8.0:
			_pick_new_wander()
		else:
			velocity = to_target.normalized() * speed
			_update_walk_anim(velocity)
	else:
		velocity = Vector2.ZERO
	move_and_slide()

func _pick_new_wander() -> void:
	var rnd := Vector2(randf_range(-1,1), randf_range(-1,1)).normalized()
	_wander_target = _home_pos + rnd * wander_range

func _update_walk_anim(v: Vector2) -> void:
	if not anim: return
	if v == Vector2.ZERO:
		if anim.has_animation("idle"): anim.play("idle")
		return
	if abs(v.x) > abs(v.y):
		if v.x > 0:
			anim.flip_h = false
			if anim.has_animation("walk_right"):
anim.play("walk_right")
		else:
			anim.flip_h = true
			if anim.has_animation("walk_right"):
anim.play("walk_right")
```

```
        else:
                if v.y < 0 and anim.has_animation("walk_up"):
                        anim.flip_h = false; anim.play("walk_up")
                elif anim.has_animation("walk_down"):
                        anim.flip_h = false; anim.play("walk_down")

func take_damage(amount: int, from: Node) -> void:
        super.take_damage(amount, from)
        if is_dead: return
        # Flip hostile if attacked
        is_hostile = true
        is_friendly = false
        set_meta("is_friendly", false)
        # TODO: optional: chase AI later
```

InteractableObject.gd

```
# res://scripts/InteractableObject.gd
extends CharacterBody2D

@onready var anim: AnimatedSprite2D = $AnimatedSprite2D

# Generic flags/fields (covers Door, Chest, ItemPickup)
@export var is_destructible: bool = false
@export var is_item: bool = false
@export var object_type: String = "" # "Door","Chest","Item"
@export var is_locked: bool = false
@export var required_key_type: String = "" # "yellow","red"
@export var is_open: bool = false
@export var is_chest_open: bool = false
@export var chest_message: String = "You opened the chest."
@export var pickup_message: String = "Picked up an item."

func _ready() -> void:
        set_collision_layer_value(5, true) # interactable
        set_collision_mask_value(2, true)  # player can overlap Area2D
        GameManager.register_entity(self, "interactable")
        if anim and anim.has_animation("item_idle") and is_item:
                anim.play("item_idle")

func on_interact(by: Node) -> void:
        match object_type:
                "Door":
                        _handle_door(by)
                "Chest":
                        _handle_chest(by)
                "Item":
                        _handle_item(by)
                _:
                        print("Interacted with object:", name)
```

```
func _handle_door(by: Node) -> void:
	if is_locked:
		# check inventory for key
		if by is BaseEntity and (by as
BaseEntity).inventory.has(required_key_type):
			is_locked = false
		else:
			# door_locked_shake if available
			if anim and anim.has_animation("door_locked_shake"):
				anim.play("door_locked_shake")
			print("Door is locked. Need key:", required_key_type)
			return

	is_open = true
	if anim:
		if anim.has_animation("door_open"): anim.play("door_open")
	print("Door opened.")

func _handle_chest(by: Node) -> void:
	if is_locked:
		if by is BaseEntity and (by as
BaseEntity).inventory.has(required_key_type):
			is_locked = false
		else:
			print("Chest is locked. Need key:", required_key_type)
			return
	is_chest_open = true
	if anim and anim.has_animation("chest_open"):
		anim.play("chest_open")
	print(chest_message)
	# TODO: add items to player's inventory here (if desired)

func _handle_item(by: Node) -> void:
	if by is BaseEntity:
		var p := by as BaseEntity
		if p.inventory.size() < p.inventory_slots:
			p.inventory.append(name) # or a typed ID
			print(pickup_message)
			queue_free()
		else:
			print("Inventory full.")
```

Animations:
Door: `door_closed`, `door_open`, `door_locked_shake`
Chest: `chest_closed`, `chest_open`
Item: `item_idle`
Player/NPC: `idle`, `walk_up`, `walk_down`, `walk_right` (flip_h for left), `attack_up`, `attack_down`, `attack_right`, `death`. Also set a custom “initial facing” simply by toggling `flip_h` or playing the appropriate idle once in `_ready()`.

---

4) Scene node structures (how to build each)

Player.tscn

```
Player (CharacterBody2D)  [Player.gd]
├─ AnimatedSprite2D
├─ CollisionShape2D
└─ InteractionArea (Area2D)
   └─ CollisionShape2D
```

1. Attach `Player.gd`.
2. Set **CollisionShape2D** to fit your sprite.
3. On `AnimatedSprite2D`, add animations named as above and set the initial one to `idle` (Autoplay).
4. Add a **Camera2D** as child of Player, check **Current** = true for simple following. (Alternative: separate camera with smoothing script, later.)

NPC.tscn

```
NPC (CharacterBody2D)  [NPC.gd]
├─ AnimatedSprite2D
├─ CollisionShape2D
└─ InteractionArea (Area2D)
   └─ CollisionShape2D
```

- Set `ai_type` to `"wander"` (default) or `"idle"`/`"patrol"`.
- `is_friendly = true` to start. They'll become hostile on damage.
- Give it the same animation set as Player (you can use fewer if you like).

Door.tscn / Chest.tscn / ItemPickup.tscn

```
Door (CharacterBody2D) [InteractableObject.gd]
├─ AnimatedSprite2D
├─ CollisionShape2D (RectangleShape2D)
└─ InteractionArea (Area2D)
   └─ CollisionShape2D
```

- Set `object_type` = `"Door"` (or `"Chest"` / `"Item"`).
- For doors/chests, set `is_locked` and `required_key_type` as needed (e.g., `"yellow"`, `"red"`).

- For items, set `is_item = true`, name it `"yellow"`/`"red"`/`"sword"` to match inventory keys.

---

5) Build Level_1 and first run

1. Create `Level_1.tscn` (Node2D or TileMap as root).
2. Instance **Player.tscn**, **NPC.tscn**, any **Door/Chest/Item** scenes into Level_1.
3. Set **Level_1.tscn** as **Main Scene** (Project > Project Settings > Run).
4. Run. Your **GameManager** will print how many players/NPCs/interactables loaded. Press **WASD** to move, **Space + direction** to attack (no direction = no attack), **E** to interact, **K** to toggle killer mode (lets you attack friendlies), **Esc** to pause/unpause.

---

6) Notes on your requirements (mapped)

- **4-directional attacks** with real hitbox; **no direction = no attack** ☑
- **Friendly NPCs cannot be attacked** unless **killer mode** is on; hostile ignores restriction ☑
- **NPCs auto-register** and flip **hostile when attacked**; simple **wander** AI provided ☑
- **Doors/Chests/Items** share one **InteractableObject** script; **E** to interact/pick up ☑
- **Auto-registering entities** and **count report** on load via **GameManager** ☑
- **Dialogue system**: intentionally **not enabled**—leave it commented for later (hook is `dialogue_text` on NPC; add a UI later). ☑
- **Animations lists** and **node structures** match your spec ☑

---

7) Where to expand next (commented hooks)

- **NPC chase/aggro**: in `NPC.gd`, if `is_hostile`, path toward nearest player.
- **Patrol**: walk between `patrol_points`.
- **Inventory UI**: show `inventory` array; enforce item types (keys, sword, potions).

- **Damage numbers/knockback**: add particles/tween.
- **Dialogue UI**: read `dialogue_text` and show a panel (currently just a data field).

If you want, I can next wire a super-simple DialoguePanel and a chest-inventory UI, but the above gets you moving, attacking, interacting, and counting—all per your plan.