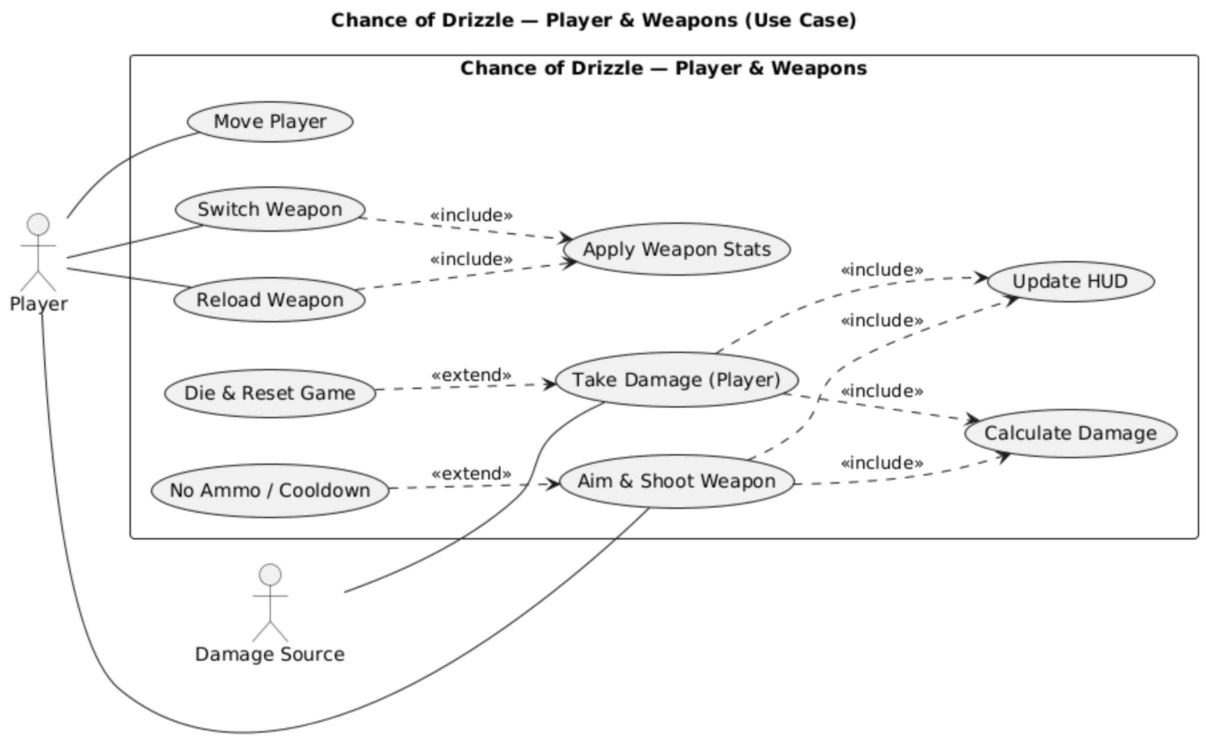
Name: Sebastjan Bas Mark ______________________/50

1. Brief introduction __/3

The Player & Weapons feature establishes the playable core of Chance of Drizzle as a roguelike shooter. It delivers responsive movement and aiming, health tracking, and a death reset loop, plus a basic ranged weapon system with predictable, scalable damage. By making the game controllable, and restart-able, this feature forms the MVP foundation on which the rest of the project enemy behavior and scaling, items/upgrades, HUD, and stage progression can be built and tested.

2. Use case diagram with scenario __14

Use Case Diagrams



Scenarios

Name: Move Player

Summary: The player moves the character around the map.

Actors: Player

Preconditions: Calculator has been initialized.

Basic sequence:

Step 1: Player presses a movement input

Step 2: System samples input and computes intended direction/velocity

Step 3: System updates player position, respecting physics and collisions

Step 4: System updates animation/state as needed.

Exceptions:

Step 3: Intended move intersects solid/forbidden space -> block movement and keep last valid position

Post conditions: Player position/animation reflect inputs

Priority: 1 (must have)

ID: C01

Name: Aim & Shoot Weapon

Summary: The player fires the equipped weapon to damage a target

Actors: Player

Preconditions: Run is active, weapon equipped.

Basic sequence:

Step 1: Player presses "Fire"

Step 2: System checks ammo and cooldown status

Step 3: System spawns projectile or applies hitscan

Step 4: System includes **Calculate Damage** to determine damage values

Step 5: System applies damage to the hit target and resolves on-hit effects.

Step 6: System includes **Update HUD** (e.g. ammo count, crosshair feedback – recoil).

Exceptions:

Step 2: No ammo or weapon on cooldown -> trigger **No Ammo / Cooldown** – do not fire.

Step 5: No valid target (miss) -> no damage applied.

Post conditions: If a target was hit, its HP is reduced and combat state updates

Priority: 1 (must have)

ID: C02

Name: Switch Weapon

Summary: The player switches to a different weapon slot

Actors: Player

Preconditions: Player has at least one alternate weapon available

Basic sequence:

Step 1: Player requests weapon switch (hotkey or scroll)

Step 2: System equips the selected weapn

Step 3: System includes **Apply Weapon Stats** – recompute derived values

Step 4: System includes **Update HUD** (weapon icon, ammo, fire rate, etc.)

Exceptions:

Step 1: No other weapon available -> ignore request and show hint

Step 2: Switch disallowed (if player stunned or any other restriction) -> ignore and show hint

Post conditions: Selected weapon becomes active with correct stats and HUD
Priority: 2 (essential)
ID: C03

Name: Reload Weapon
Summary: The player reloads the equipped weapon
Actors: Player
Preconditions: Weapon supports reload and is not already full
Basic sequence:

Step 1: Player presses "Reload"
Step 2: System plays reload sequence and updates internal ammo state
Step 3: System includes **Apply Weapon State**
Step 4: System includes **Update HUD** – new ammo count

Exceptions:

Step 1: Magazine already full -> ignore and show hint
Step 2: Reaload interrupted (player cancels or takes action that cancels reload)

Post conditions: Weapon ammo is replenished per weapon rules, HUD reflects current ammo
Priority: 2 (essential)
ID: C04

Name: Calculate Damage
Summary: Compute damage output considering weapn, upgrades, and target defenses
Actors: Invoked by parent use case
Preconditions: Caller provides attacker stats, weapon data, modifiers, and target defenses.
Basic sequence:

Step 1: Receive inputs (base damage, multipliers, crit, elemental/status, target, armor/resistance)
Step 2: Compute base damage, apply multipliers/additives in defined order of operations.
Step 3: Apply target mitigations (armor, resistance, shields)
Step 4: Clamp to valid range (>= 0, <= MAX)
Step 5: Return final damage value to caller

Exceptions:

Step 1,2: Missing/invalid parameters -> use safe defaults or return 0
Step 4: Overflow/underflow

Post conditions: A deterministic damage value is returned to the caller
Priority: 1 (must have)
ID: C05

Name: Take Damage
Summary: The player receives and processes incoming damage from an external source
Actors: Damage Source
Preconditions: Player health > 0, a validated hit is registered
Basic sequence:

> **Step 1:** System receives a hit event against the player
> **Step 2:** System includes **Calculate Damage** to determine incoming damage value.
> **Step 3:** System subtracts damage from player HP and applies on-hit effects (e.g., knockback, status).
> **Step 4:** System includes **Update HUD** (HP bar, hit flash).
> **Step 5:** If HP ≤ 0 → trigger **Die & Reset Game**

Exceptions:

> **Step 2:** Player has invulnerability (i-frames, shield) → treat damage as 0, still show minimal feedback if desired.
> **Step 3:** Damage reduced to below 0 after mitigation → clamp to 0.

Postconditions: Player HP updated, death workflow initiated if HP ≤ 0.
Priority: 1 (must have)
ID: C06

Name: Die & Reset Game
Summary: Handle player death and reset the run.
Actors: Triggered by C06 as an extension
Preconditions: Player HP ≤ 0.
Basic sequence:

> **Step 1:** System disables player input and marks run as over.
> **Step 2:** System records end-of-run data (optional: stats/score).
> **Step 3:** System clears or resets transient state (stage progress, temporary upgrades per design).
> **Step 4:** System respawns player at starting state (Stage 1 or hub) with baseline equipment/stats.
> **Step 5:** System includes **Update HUD** (reset values).

Exceptions:

> **Step 4:** Persisted unlocks/meta-progress apply on spawn (if present) → load them.

Postconditions: New run is ready, player is alive with baseline state.
Priority: 1 (must have)
ID: C07

Name: No Ammo / Cooldown
Summary: Handle attempts to fire when ammo is zero or weapon is cooling down.
Actors: Triggered by C02 as an extension
Preconditions: Player pressed Fire, weapon has 0 ammo or cooldown remaining.
Basic sequence:

> **Step 1:** System blocks firing action (no projectile / no hitscan).
> **Step 2:** System plays "dry fire"/cooldown feedback (sound/animation).
> **Step 3:** System includes **Update HUD** (ammo=0 or cooldown indicator).

Exceptions:

> **Step 1:** If weapon allows "charge without ammo" (design-specific) → allow animation but still no shot.

Postconditions: No shot is fired, player receives clear feedback on why.
Priority: 2 (essential)
ID: C08

Name: Apply Weapons Stats
Summary: Recompute and apply derived weapon stats after a change (switch/reload).
Actors: Invoked by C03/C04
Preconditions: A weapon switch or reload has been requested.
Basic sequence:

> **Step 1:** Read base weapon profile (damage, fire rate, spread, magazine, reload time).
> **Step 2:** Apply player/upgrades/modifiers that affect weapon stats.
> **Step 3:** Update equipped weapon instance with new derived values.
> **Step 4:** System includes **Update HUD** (display updated stats/ammo).

Exceptions:

> **Step 2:** Conflicting modifiers → resolve by priority order (documented rules).

Postconditions: Equipped weapon's runtime stats match current conditions, HUD reflects changes.
Priority: 2 (essential)
ID: C09

Name: Update HUD

Summary: Refresh on-screen information relevant to the player and weapons.

Actors: Invoked by parent use cases

Preconditions: Caller provides the specific fields to refresh (e.g., ammo, HP, weapon icon).

Basic sequence:

> **Step 1:** Receive update request (what changed).
>
> **Step 2:** Recompute displayed values and text/icons.
>
> **Step 3.** Redraw affected HUD elements.

Exceptions:

> **Step 2:** Missing data for a field → leave previous value and log (optional).

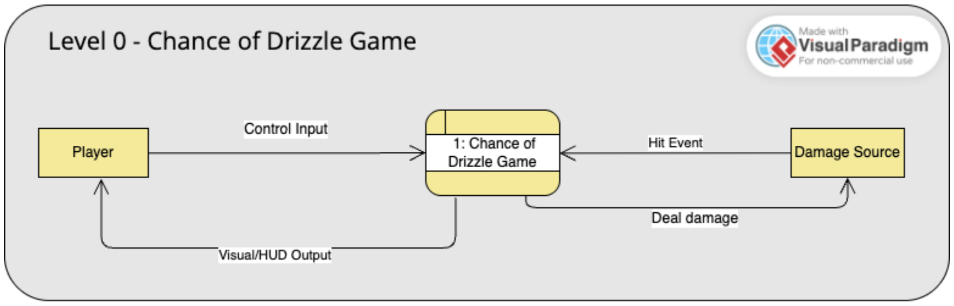
Postconditions: HUD reflects current player/weapon state.

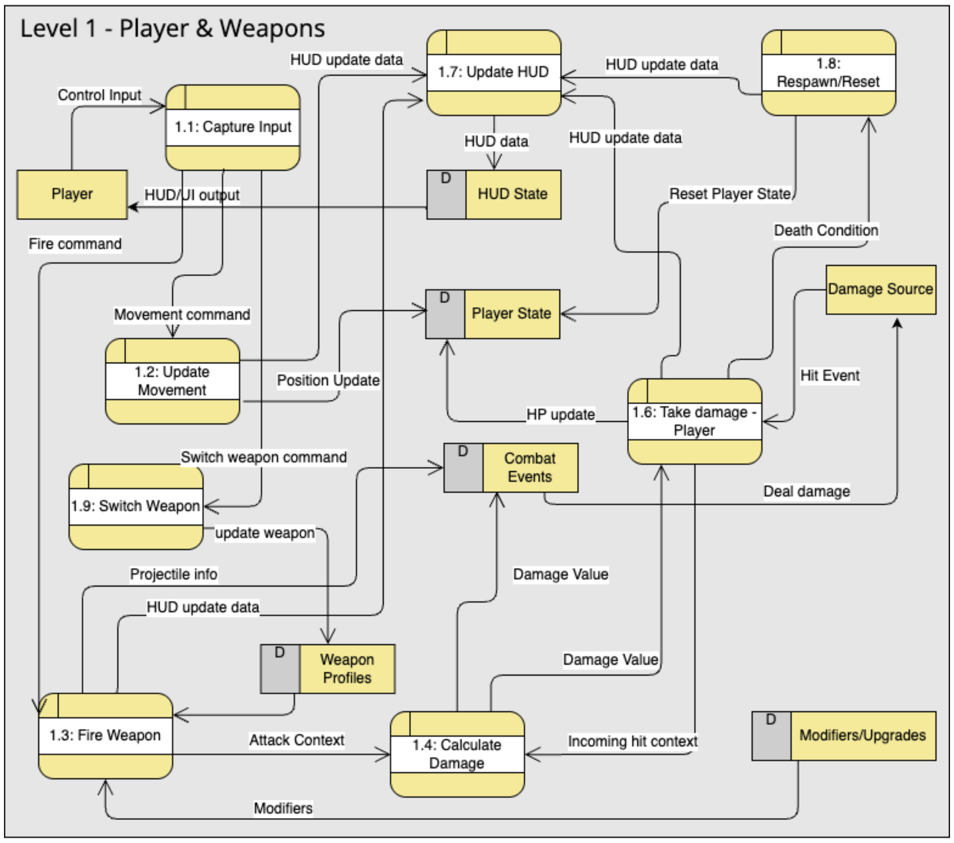
Priority: 2 (essential)

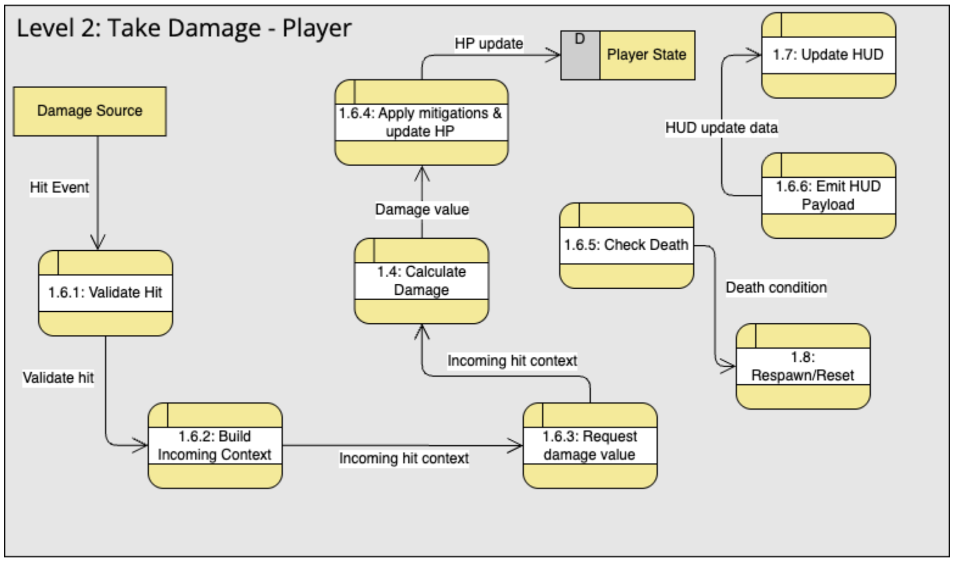
ID: C10

3. Data Flow diagram(s) from Level 0 to process description for your feature ______14

Data Flow Diagrams:


Level 0 - Chance of Drizzle Game
Made with Visual Paradigm For non-commercial use
Player
Control Input
1: Chance of Drizzle Game
Hit Event
Damage Source
Deal damage
Visual/HUD Output


Level 1 - Player & Weapons
HUD update data
1.7: Update HUD
HUD update data
1.8: Respawn/Reset
Control Input
1.1: Capture Input
HUD data
HUD update data
Player
HUD/UI output
D
HUD State
Reset Player State
Fire command
Death Condition
Damage Source
Movement command
D
Player State
1.2: Update Movement
Position Update
Hit Event
HP update
1.6: Take damage - Player
Switch weapon command
D
Combat Events
1.9: Switch Weapon
Deal damage
update weapon
Projectile info
Damage Value
HUD update data
D
Weapon Profiles
Damage Value
1.3: Fire Weapon
Attack Context
1.4: Calculate Damage
Incoming hit context
D
Modifiers/Upgrades
Modifiers


Level 2: Take Damage - Player
HP update
D
Player State
1.7: Update HUD
Damage Source
1.6.4: Apply mitigations & update HP
HUD update data
Hit Event
Damage value
1.6.6: Emit HUD Payload
1.6.5: Check Death
1.6.1: Validate Hit
1.4: Calculate Damage
Death condition
1.8: Respawn/Reset
Validate hit
Incoming hit context
1.6.2: Build Incoming Context
Incoming hit context
1.6.3: Request damage value

Process Descriptions

Level-1 Process Descriptions

1.1 — Capture Input

Purpose. Normalize raw player controls and fan them out as commands to downstream processes.
Inputs. Player → Control Input (keys, mouse, gamepad axes/buttons).
Outputs. → 1.2 Movement Command → 1.3 Fire Command → 1.9 Switch Weapon Command → 1.7 HUD Update.
Reads/Writes. (none)
Preconditions. Run is active, input device detected.

Logic.

```
LOOP each frame:
  sample raw axes/buttons
  apply deadzones & normalize ([-1..1] axes, radians aim vector)
  build Movement Command (dir, sprint, jump flags)
  IF Fire pressed THEN send Fire Command to 1.3
  IF weapon-cycle pressed THEN send Switch Weapon Command to 1.9
  emit HUD Update Payload if needed (e.g., interaction prompts)
END LOOP
```

1.2 — Update Movement

Purpose. Move the player and resolve collisions.
Inputs. 1.1 → Movement Command.
Outputs. → D1 Position Update → 1.7 HUD Update (e.g., sprint/stamina indicators).
Reads/Writes. Read D1 Player State (pos/vel/caps) write D1 Position Update.
Preconditions. Valid player entity exists.

Logic.

```
derive desired velocity from Movement Command and Player State
apply gravity & friction
perform collision sweep → resolve penetration
update Player State (position, velocity, grounded flag, facing/aim)
emit HUD Update Payload if move state affects UI (e.g., stamina)
```

Postconditions. Player position/velocity updated, collisions respected.

1.3 — Fire Weapon

Purpose. Validate a fire request, create a projectile or hitscan event, and manage ammo/cooldown.
Inputs. Player → Fire Command (from 1.1).
Outputs. → 1.4 Attack Context (request), ← 1.4 Damage Value (response), → D5/Combat Events or

external Enemy system: Damage Packet, → 1.7 HUD Update.
Reads/Writes. Read D2 Weapon Profiles, read/write D1 Player State (ammo, cooldown). Optionally write D5 Combat Events.
Preconditions. Equipped weapon exists in D1, profile exists in D2.

Decision table — Validate fire

Cond.	Ammo>0	Cooldown=0	Weapon functional	Outcome
F1	Yes	Yes	Yes	ALLOW (continue)
F2	No	*	*	BLOCK (emit HUD “no ammo”)
F3	*	No	*	BLOCK (emit HUD “cooling”)
F4	*	*	No	BLOCK (emit HUD “disabled”)

Logic.

```
on Fire Command:
  evaluate F1..F4
  IF BLOCK THEN emit HUD Update Payload and EXIT
  build Attack Context from D2 (weapon), D1 (aim/pos), D3 (modifiers if your
diagram routes them here)
  Damage Value := request from 1.4 Calculate Damage(Attack Context)
  create projectile OR apply hitscan; package Damage Packet
  send Damage Packet to Enemy/World (or write to D5 Combat Events)
  decrement ammo; start cooldown; write back to D1
  emit HUD Update Payload (ammo, reticle flash, cooldown)
```
Postconditions. A combat event was produced (or correctly blocked) and weapon state updated.

1.4 — Calculate Damage

Purpose. Return a deterministic damage value from either an attack or an incoming hit.
Inputs. From 1.3: Attack Context from 1.6: Incoming Hit Context.
Outputs. To caller: Damage Value (optional) log/telemetry.
Reads/Writes. Read D2 Weapon Profiles and D3 Modifiers/Upgrades as needed.
Preconditions. Context includes base stats and target/attacker properties.

Logic.

```
extract base (weaponBase or attackerBase), multipliers, additive bonuses
compute critChance/critMult if present
raw := base * Π(multipliers) + Σ(additives)
IF crit proc THEN raw := raw * critMult
apply target mitigations (armor/resist/shields) if in context
return clamp(raw, 0, MAX_DAMAGE)
```
Postconditions. Caller receives a single scalar value suitable for HP subtraction or effect scaling.

1.6 — Take Damage — Player

Purpose. Process an external hit on the player and update HP, death, and UI.
Inputs. Damage Source → Hit Event, exchanges with 1.4 for Incoming Hit Context / Damage Value.
Outputs. → D1 HP Update → 1.8 Death Condition (if HP ≤ 0) → 1.7 HUD Update.
Reads/Writes. Read/write D1 Player State.
Preconditions. Player entity alive (HP>0) at event time.

Logic.

```
validate Hit Event (team/phase/invulnerability)
build Incoming Hit Context from event + Player State
Damage Value := 1.4 Calculate Damage(Incoming Hit Context)
HP := max(0, HP - Damage Value); write HP back to D1
emit HUD Update Payload (HP bar, hit flash)
IF HP <= 0 THEN send Death Condition to 1.8
```

Postconditions. Player HP reflects the hit, death is escalated if needed.

1.7 — Update HUD

Purpose. Aggregate changes and render HUD/UI output to the player.
Inputs. From 1.2/1.3/1.6/1.8: HUD Update Payload (what changed).
Outputs. → D4 HUD State → Player HUD/UI Output.
Reads/Writes. Read D1/D2/D3 as needed to format, write D4.
Preconditions. At least one payload or periodic refresh tick.

Logic.

```
collect pending HUD Update Payloads this frame
merge and normalize (dedupe same-field updates)
query D1/D2/D3 for exact values to display
write composed result to D4 (HUD State)
send rendered HUD/UI Output to Player
```

Postconditions. On-screen info matches current game state.

1.8 — Respawn/Reset

Purpose. Reset the run after death and make the game ready to play again.
Inputs. From 1.6: Death Condition.
Outputs. → D1 Reset Player State (baseline stats/position/weapon) → 1.7 HUD Update.
Reads/Writes. Write D1 optionally read meta-progress data if you have it.
Preconditions. Player HP ≤ 0.

Logic.

```
disable player input & mark run over
(optional) record end-of-run stats
```

```
reset D1 Player State to baseline (HP full, pos spawn, starter weapon, clear
temp modifiers)
clear transient stores (timers, status effects) as needed
emit HUD Update Payload (reset bars, icons, stage indicator)
re-enable input; mark run active
```

Postconditions. New run is initialized UI reflects reset.

1.9 — Switch Weapon

Purpose. Equip a different weapon and update derived stats.

Inputs. From 1.1: Switch Weapon Command.

Outputs. → D1 Update Weapon (equipped slot/id) → 1.7 HUD Update.

Reads/Writes. Read D2 Weapon Profiles, write D1 Player State.

Preconditions. Player owns at least one alternative weapon.

Logic.

```
resolve target slot/weapon from command
IF unavailable THEN emit HUD tip and EXIT
write equipped weapon id/slot into D1
(option) recompute derived weapon stats from D2/D3 and store in D1
emit HUD Update Payload (weapon icon, ammo, fire rate)
```

Postconditions. Selected weapon is active, HUD shows new weapon info.

Level-2 Child Process Descriptions (for 1.6 Take Damage — Player)

1.6.1 — Validate Hit

Purpose. Confirm the hit applies now (e.g., not during i-frames).

Inputs. Damage Source → Hit Event read D1 (invulnerability/shield states).

Outputs. → 1.6.2 Validated Hit (or drop event).

Logic.

```
IF team/faction invalid OR player is invulnerable OR shielded THEN DROP
ELSE pass Validated Hit to 1.6.2
```

1.6.2 — Build Incoming Context

Purpose. Assemble the data 1.4 needs to compute damage.

Inputs. Validated Hit + D1 Player State (resistances, armor).

Outputs. → 1.6.3 Incoming Hit Context.

Logic.

```
context := { attackerType, basePower, damageType, playerResistances, armor,
buffs/debuffs }
```

1.6.3 — Request Damage Value

Purpose. Call the shared calculator and return its result.
Inputs/Outputs. → 1.4 Incoming Hit Context ← 1.4 Damage Value → 1.6.4 Damage Value.
Logic. "Pass-through" orchestration.

1.6.4 — Apply Mitigations & Update HP

Purpose. Subtract the damage from HP and persist the result.
Inputs. Damage Value read/write D1.
Outputs. → D1 HP Update → 1.6.6 HUD Update Payload → 1.6.5 HP Status.
Logic.

```
HP := max(0, HP - Damage Value)
write HP to D1
emit HUD Update Payload (HP delta, effects)
send HP Status to 1.6.5
```

1.6.5 — Check Death

Purpose. Decide if death processing is needed.
Inputs. HP Status (current HP).
Outputs. → 1.8 Death Condition (if HP ≤ 0).
Logic.

```
IF HP <= 0 THEN send Death Condition
```

1.6.6 — Emit HUD Payload

Purpose. Forward UI changes caused by damage.
Inputs. Signals from 1.6.4/1.6.5 (HP change, death).
Outputs. → 1.7 HUD Update Payload.
Logic. Create a minimal, deduplicated payload (HP bar, damage flash, death icon).

4. Acceptance Tests _______9

All tests are automatable with the Unity Test Framework (EditMode for pure functions, PlayMode with a test scene + PhysicsScene for movement/weapon flows). We assert on world/state values (positions, HP, ammo, cooldowns, HUD flags) rather than keypresses or pixels.

Legend:

- Step S# refers to the **Basic Sequence** step.

C01 – Move Player

Test ID	UC/Step	Preconditions	Input / Action	Expected Result (Pass Criteria)
AT-MOVE-01	Use Case C01 S1–S3	Player at (0,0); ground; no obstacles	Issue Movement Command +X for 1.0 s	`x ≈ MOVE_SPEED ± ε`, `y ≈ 0`; animation/state updated
AT-MOVE-02 (Boundary—Collision)	Use Case C01 S3	Player at x=4.9; wall at x=5	Issue Movement Command +X for 1.0 s	`x ≤ 5 − skinWidth` (no penetration)
AT-MOVE-03 (Bad Input Clamp)	Use Case C01 S2	—	Movement Command magnitude > 1 (e.g., (2,0))	Effective speed ≤ `MOVE_SPEED` (vector normalized)

C02 - Aim & Shoot Weapon

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result (Pass Criteria)
AT-FIRE-01	C02 S1–S6	ammo=10; cooldown=0; target present	Fire once	One projectile/hitscan event; exactly 1 Damage Packet emitted; ammo=9; cooldown ≈ `COOLDOWN`; HUD ammo updated
AT-FIRE-02 (E2a – No Ammo)	C02 S2 → Exception	ammo=0; cooldown=0	Fire once	No projectile/packet; ammo still 0; cooldown unchanged; HUD “no ammo” cue
AT-FIRE-03 (E2b – Cooldown)	C02 S2 → Exception	ammo>0; cooldown>0	Fire once	No projectile/packet; ammo unchanged; HUD cooldown cue
AT-FIRE-04 (Calculator Happy-Path)	C02 S4	weapon base=10; mod ×2; target armor −5	Fire once	DamageValue returned by C05 == 15 and within `[0, MAX_DAMAGE]`
AT-FIRE-05 (Bad Profile Safe-Default)	C02 S4	weapon profile has NaN/missing fields	Fire once	Calculator returns 0 (clamped); no crash; HUD still updates

C03 – Switch Weapon

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-SWITCH-01	C03 S1–S4	Two valid weapons	Switch to slot 2	Equipped weapon = slot 2; derived stats applied; HUD (icon/ammo) updated
AT-SWITCH-02 (Unavailable Slot)	C03 S1 → Exception	Only one weapon owned	Switch to slot 2	Equipped unchanged; HUD hint “no weapon”

C04 – Reload Weapon

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-RELOAD-01	C04 S1–S4	Mag 10/30; reserve ≥20	Reload	Mag=30; reserve decremented; HUD ammo updated
AT-RELOAD-02 (Boundary—Full Mag)	C04 S1 → Exception	Mag already full	Reload	No state change; HUD hint "full"

C05 – Calculate Damage

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-DMG-01 (Clamp Low)	C05	—	base = −10	Returns 0
AT-DMG-02 (Clamp High)	C05	—	inputs that exceed `MAX_DAMAGE`	Returns MAX_DAMAGE
AT-DMG-03 (Deterministic Crit)	C05	critChance=100%	Evaluate once	Returns `base × critMult` exactly
AT-DMG-04 (Null/Empty Context)	C05	—	context = null/empty	Returns 0; no exception

C06 – Take Damage (Player)

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-HIT-01	C06 S1–S5	HP=100; DamageValue=15	HitEvent once	HP=85; HUD HP changed; no DeathCondition
AT-HIT-02 (Invulnerability)	C06 S2 → Exception	i-frames active	HitEvent once	HP unchanged; minimal feedback allowed
AT-HIT-03 (Boundary to Death)	C06 S5 → Exception → <extend>	HP=10; DamageValue=10	HitEvent once	HP=0; DeathCondition raised to C08
AT-HIT-04 (Bad Damage)	C06 S3–S4	DamageValue < 0 or NaN	Apply	Treated as 0; HP unchanged; no crash

C07 – Update HUB

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-HUD-01 (Merge Payloads)	C07	Pending ammo & HP payloads same frame	Update once	D4/HUD shows both updates; no duplicate

				redraw
AT-HUD-02 (Missing Field)	C07	Payload references unknown key	Update once	Old value retained; warning logged; no exception

C08 – Respawn/Reset

Test ID	Use Case/Step	Preconditions	Input / Action	Expected Result
AT-RESP-01 (Full Reset)	C08	Death Condition received	Execute	D1 reset to baseline (HP full, spawn pos, starter weapon, temp mods cleared); HUD shows reset
AT-RESP-02 (Idempotent)	C08	Already at baseline	Execute twice	State remains baseline; no duplicate spawns

5. Timeline ________/10

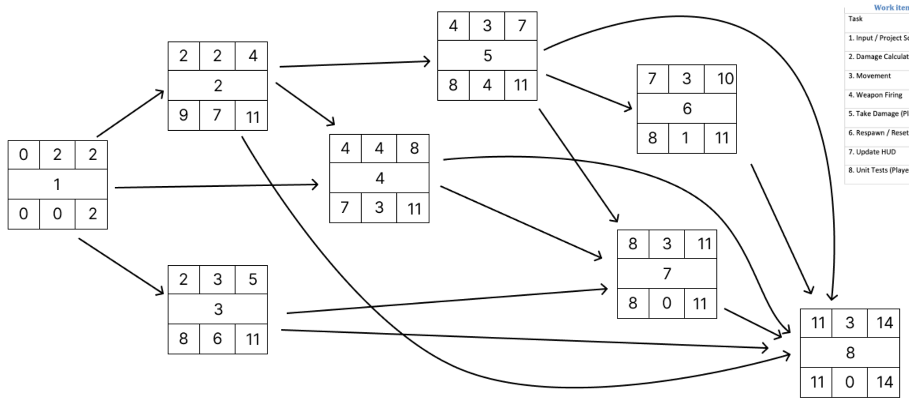
[Figure out the tasks required to complete your feature]

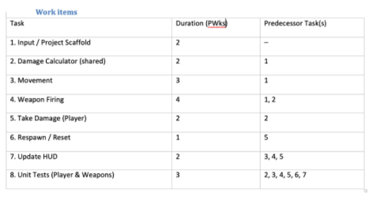
Example:

Work items

Task	Duration (PWks)	Predecessor Task(s)
1. Input / Project Scaffold	2	–
2. Damage Calculator (shared)	2	1
3. Movement	3	1
4. Weapon Firing	4	1, 2
5. Take Damage (Player)	2	2
6. Respawn / Reset	1	5
7. Update HUD	2	3, 4, 5
8. Unit Tests (Player & Weapons)	3	2, 3, 4, 5, 6, 7

Pert diagram

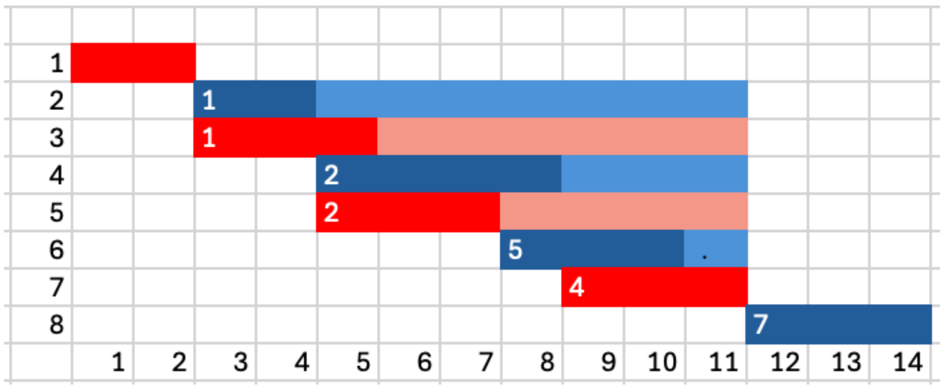




Work items

Task	Duration (PWks)	Predecessor Task(s)
1. Input / Project Scaffold	2	–
2. Damage Calculator (shared)	2	1
3. Movement	3	1
4. Weapon Firing	4	1, 2
5. Take Damage (Player)	2	2
6. Respawn / Reset	1	5
7. Update HUD	2	3, 4, 5
8. Unit Tests (Player & Weapons)	3	2, 3, 4, 5, 6, 7

Gantt timeline


1
2
3
4
5
6
7
8
1
1
2
2
5
4
7
1 2 3 4 5 6 7 8 9 10 11 12 13 14