# FINDING THE FUN IN PHYSICS

*building a physics-based game in Unity*

## INTRODUCTION

- *Who we are*
- *What we're making*
- *What this talk is about*

NINJA THUMBS

@teamninjathumbs

## WHO WE ARE

- **NINJA THUMBS**
  - Moritz Schlitter *(**ART**)*
  - Steve Salmond *(**CODE**)*
- **CONTRIBUTORS**
  - Clark Aboud *(**MUSIC**)*
    *clarkaboud.com*

# GRABITY

robot battle testing

grabitygame.com

# WHAT WE'RE MAKING

- GRABITY
  - A physics-based **arena brawler** for 2-4 players
  - Wield **grab guns** to grab and shoot objects
  - **2.5D** (3D world, most action on Z=0 plane)
  - Emphasis on **fluid movement**

## GAMEPLAY

- A quick gameplay snippet to illustrate…

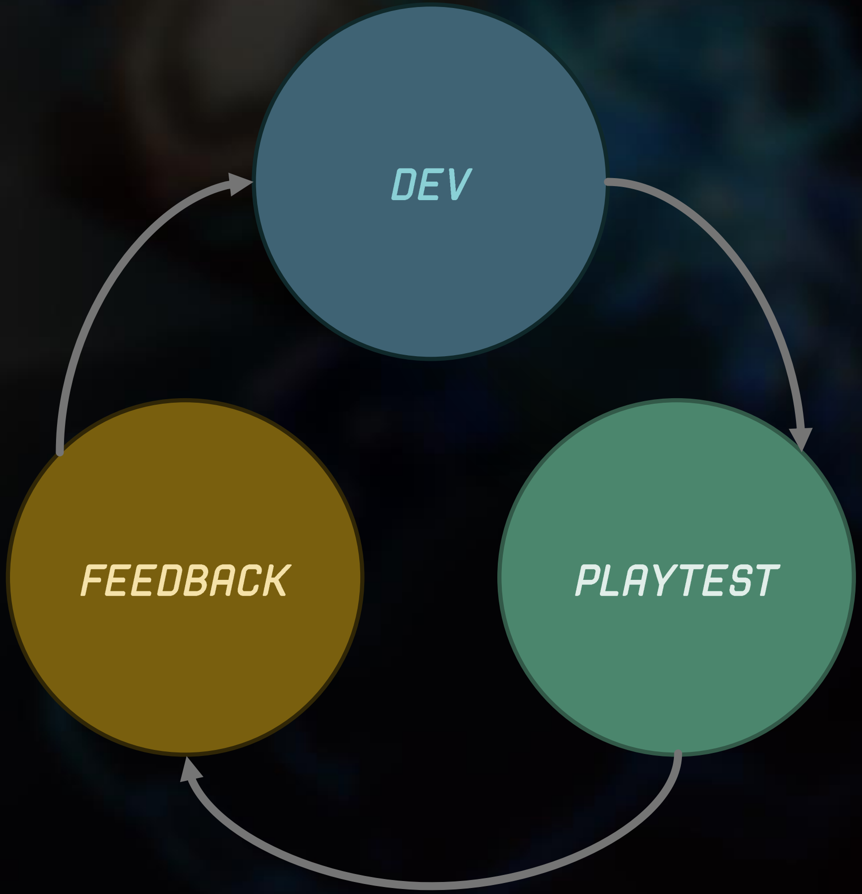
# WHAT THIS TALK IS ABOUT

## WHAT THIS TALK IS ABOUT

- *Character control with* ***physics****!*
- *Not too technical (hopefully)*

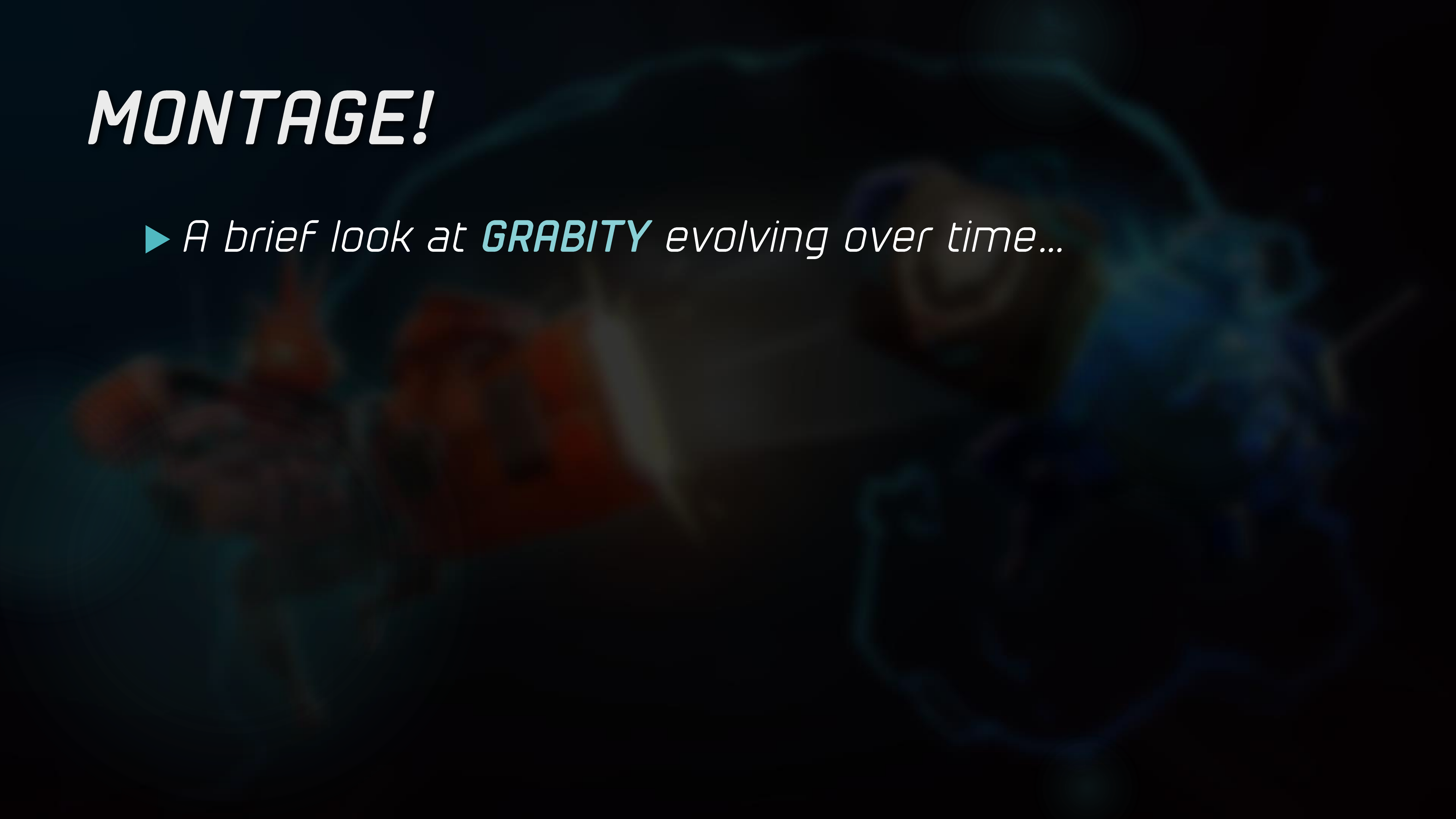
# ITERATION

- ***PROBLEM***
  - **Game feel** can be tricky in a physics environment!
- ***APPROACH***
  - ***DEVELOP*** a new mechanic
  - ***PLAYTEST*** the heck out of it
  - Get ***FEEDBACK*** from players
  - Rinse and repeat!



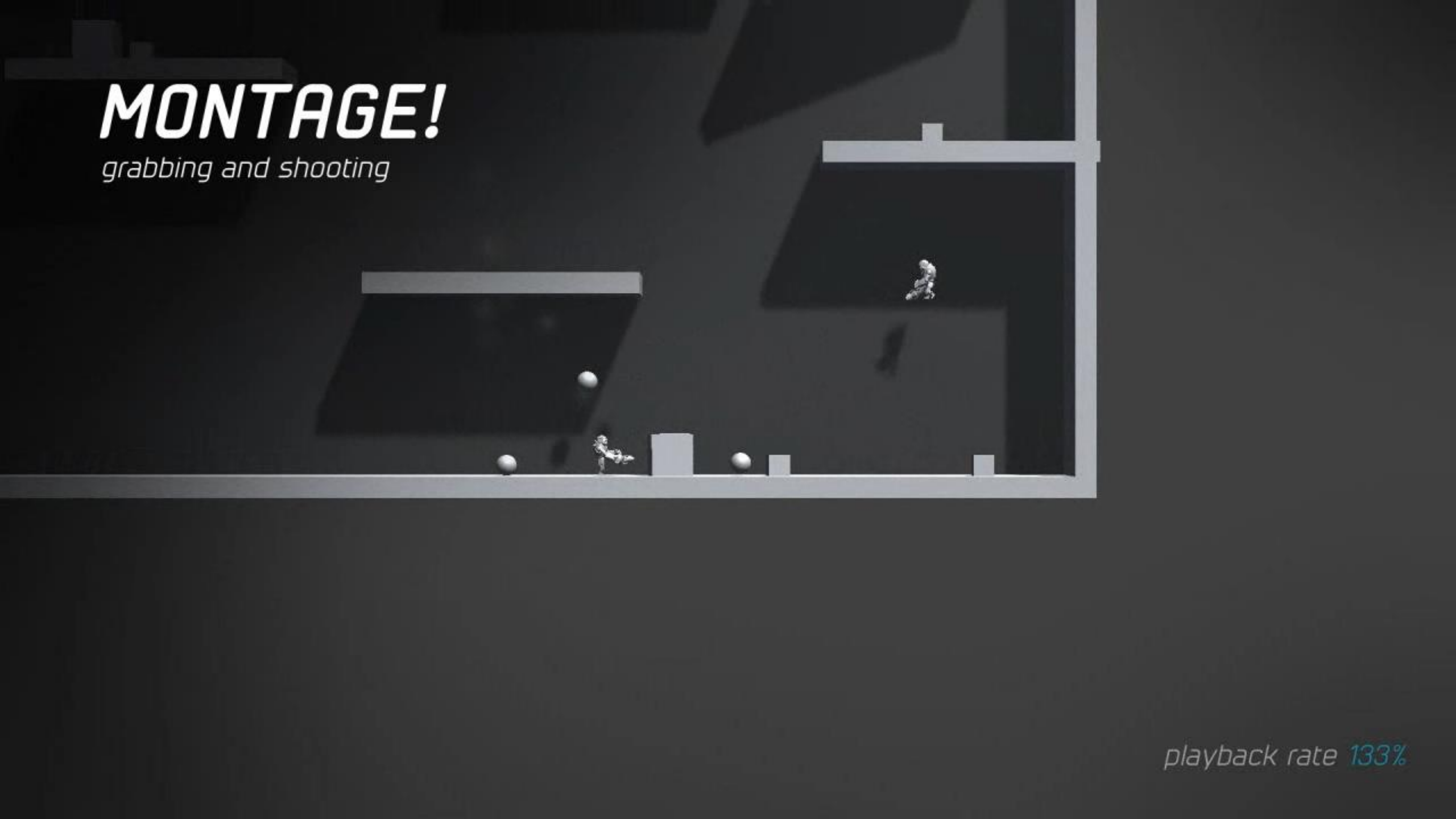
## MONTAGE!

- A brief look at **GRABITY** evolving over time…



## MONTAGE!

*grabbing and shooting*



playback rate 133%

## MECHANICS

- ITERATIVE APPROACH
  - Started with simple controls
  - Progressively added and refined mechanics
  - Let's quickly look at each mechanic in turn..

## MECHANICS

- **Movement**
- Braking
- Jumping
- Wall jumping
- Crouching
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting

## MOVEMENT

- *Convert player input into a force*
  - Get input vector (dx, dy) from controller
  - Apply axis weighting
    *e.g. (1, 0) on ground, (1, 0.1) in the air*
  - Ensure input vector length <= 1
  - Scale by a conversion factor to obtain **input force**
- *Apply force to Rigidbody each physics step*

## MOVEMENT

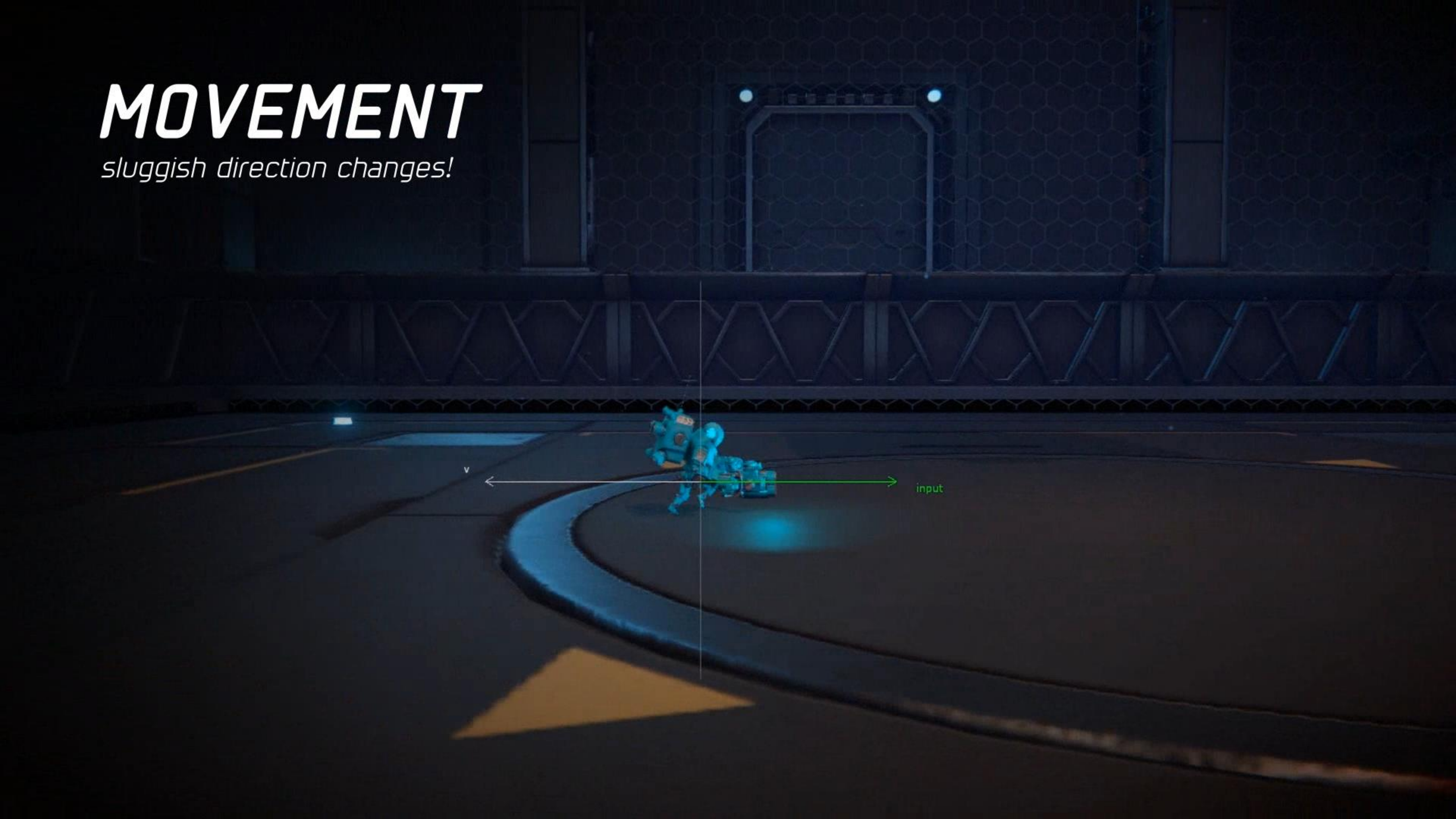
```
// Get player's weighted movement input vector.
dx = Controller.GetAxis("Horizontal") * axisWeight.x;
dy = Controller.GetAxis("Vertical") * axisWeight.y;
input = new Vector3(dx, dy, 0);

// Ensure input vector’s length is 1 or less.
if (input.magnitude > 1)
    input = input.normalized;

// Apply movement force.
force = input * InputForceScale;
Body.AddForce(force);
```

## MOVEMENT

*sluggish direction changes!*



## MOVEMENT

- ***PROBLEMS***
  - Takes ages for player to come to rest
  - Sluggish direction changes
  - Unlimited top speed

- ***SOLUTION***
  - Apply a **braking force** that opposes lateral velocity
  - At max speed, cancels **input force** completely
  - **Dynamic drag** *(0 when active input, otherwise 1)*

# MECHANICS

- Movement
- **Braking**
- Jumping
- Wall jumping
- Crouching
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting

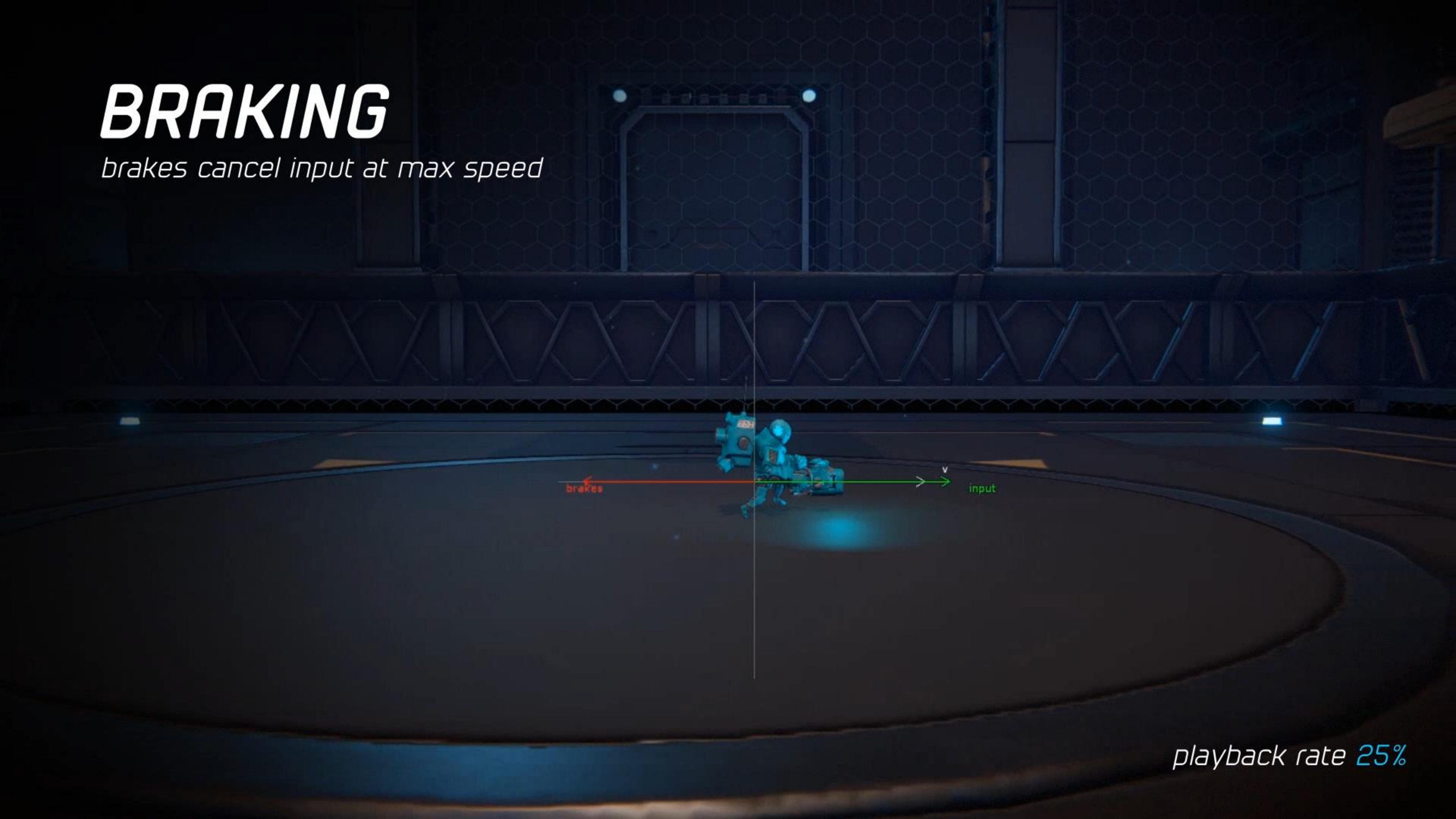
## BRAKING

```
// Compute braking factor.
speed = Vector3.Dot(velocity, right);
brakes = left * (speed / maxSpeed);

// Apply overall movement force.
force = (input + brakes) * InputForceScale;
Body.AddForce(force);
```

## BRAKING

*brakes cancel input at max speed*



*playback rate 25%*

## MECHANICS

- Movement
- Braking
- **Jumping**
- Wall jumping
- Crouching
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting

# JUMPING

- ***FIRST ATTEMPT***
  - Add a big **upwards force** to jump
  - Unpredictable results..

- ***SECOND ATTEMPT***
  - Modify **velocity** directly
  - Retain v.x, reset v.y
  - Scale up v.y based on v.x (‘running’ jumps)

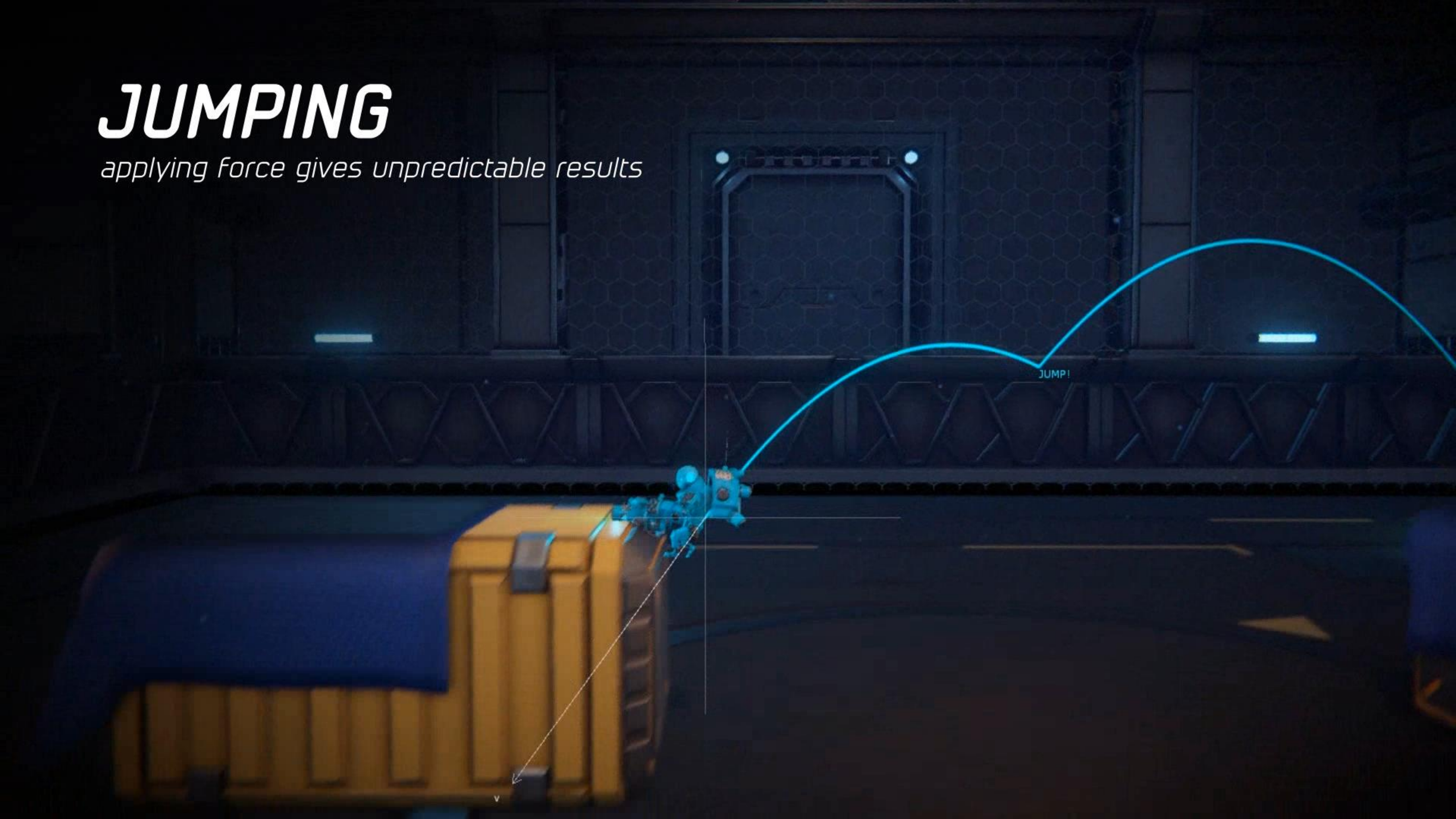
# JUMPING

```
// Zero vertical component if airborne.
v = Body.velocity;
if (!grounded)
    v.y = 0;

// Apply jump (depending on lateral speed).
lateralSpeed = Abs(Vector3.Dot(right, v));
speed = JumpSpeed.Evaluate(lateralSpeed);
Body.velocity = v + (up * speed);
```

## JUMPING

*applying force gives unpredictable results*



## MECHANICS

- Movement
- Braking
- Jumping
- **Wall jumping**
- Crouching
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting

## *WALL JUMPING*

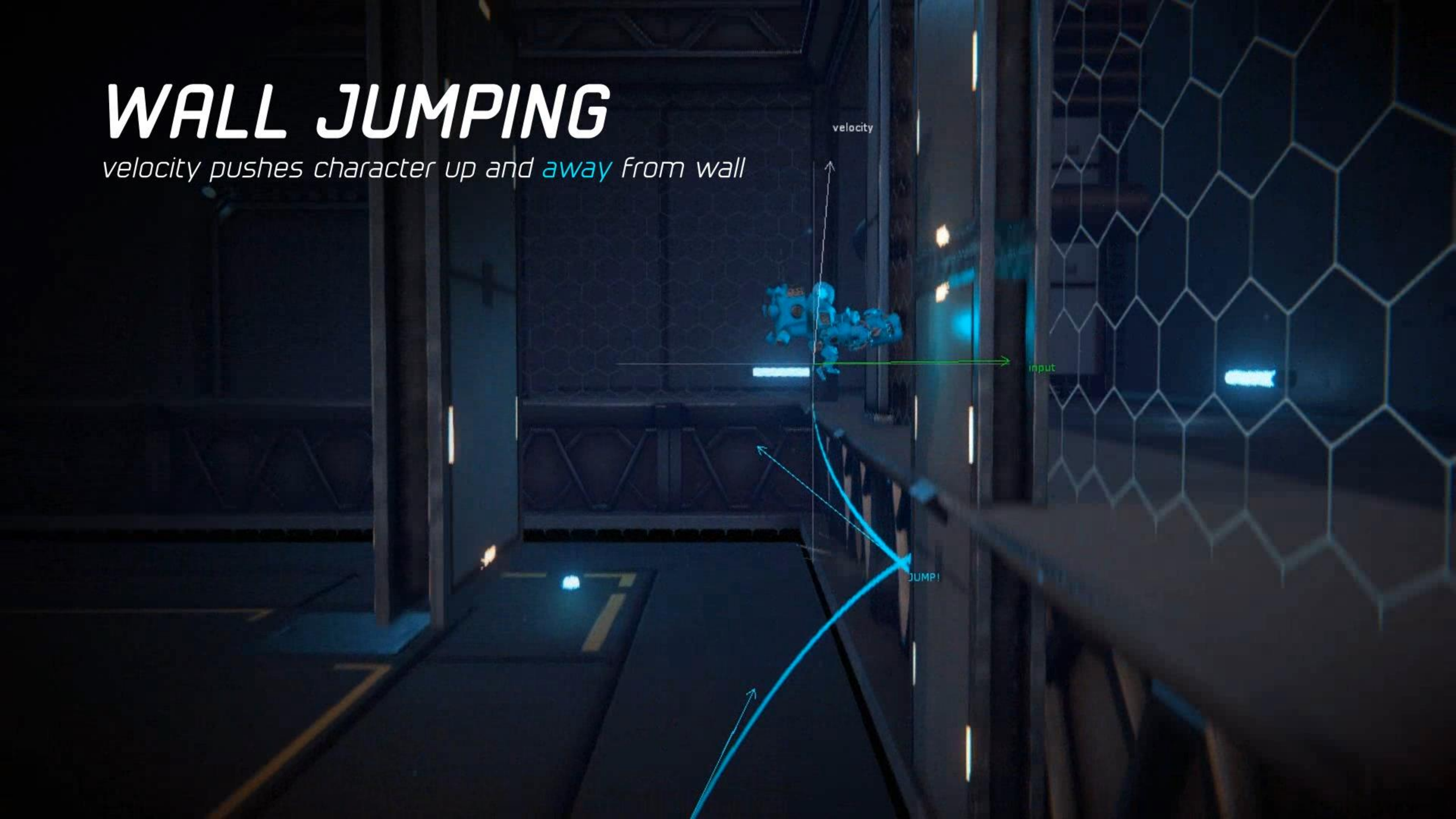
- *Use raycasts to detect wall proximity*
- *Like regular jumping, but*
  - Apply **lateral** as well as upwards **velocity**
  - Decrease **input force** on walls to slide down

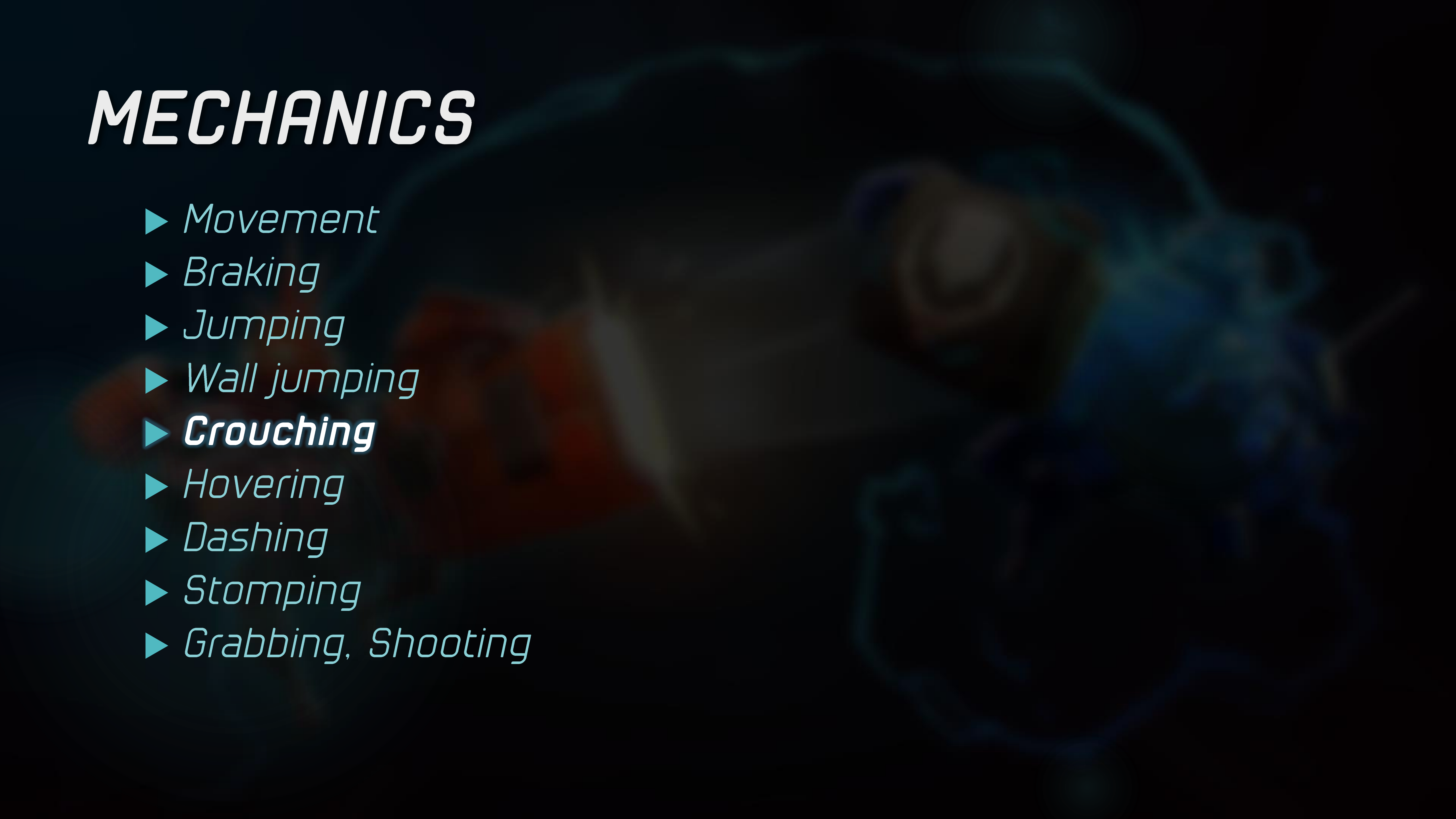
# WALL JUMPING

```
// Apply wall-jumping as needed.
if (isAgainstRightWall && !grounded)
   v += left * WallJumpSpeed;
if (isAgainstLeftWall && !grounded)
   v += right * WallJumpSpeed;
```

# WALL JUMPING

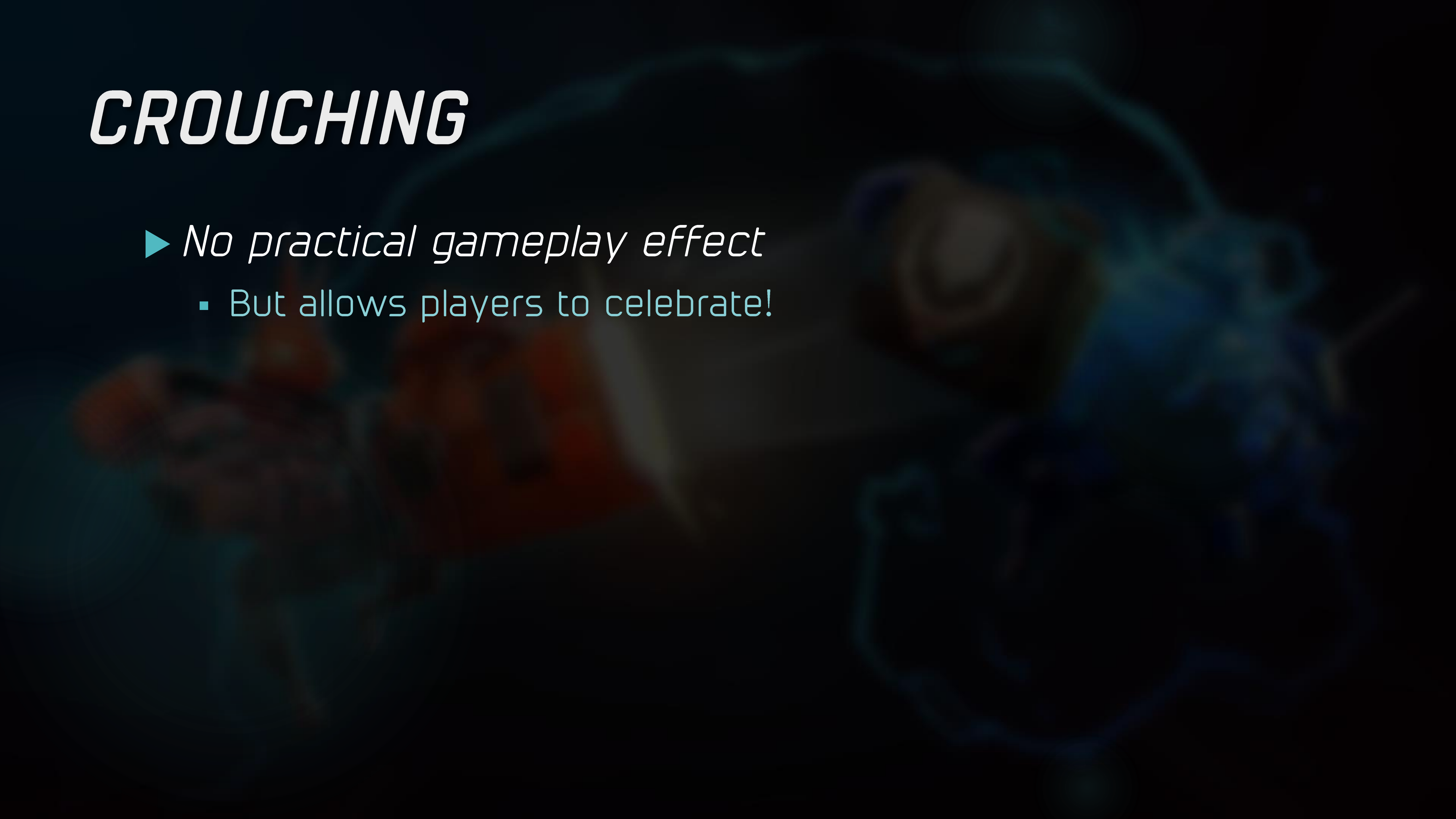
*velocity pushes character up and away from wall*





## MECHANICS

- Movement
- Braking
- Jumping
- Wall jumping
- **Crouching**
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting



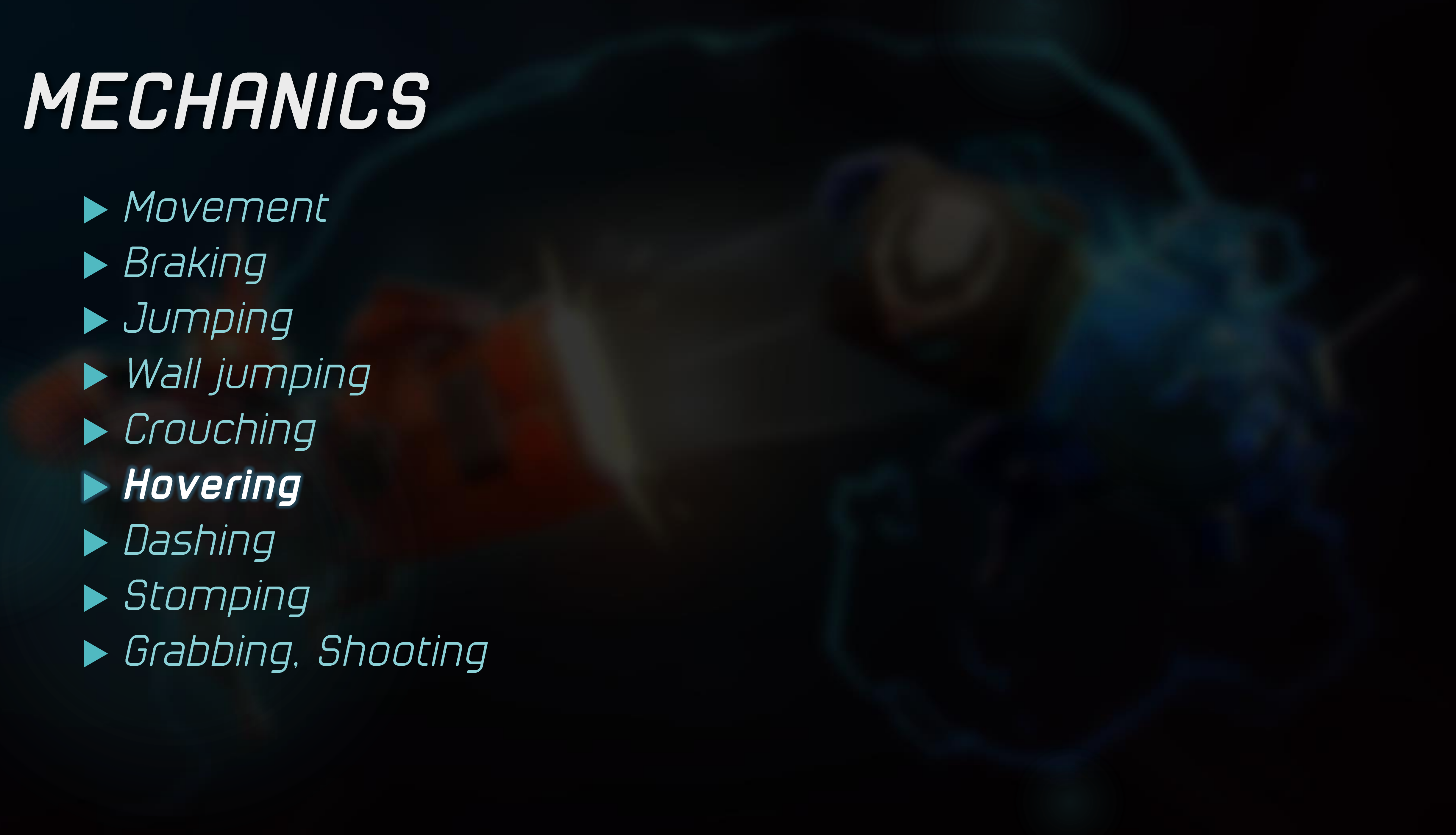
## CROUCHING

- *No practical gameplay effect*
  - But allows players to celebrate!

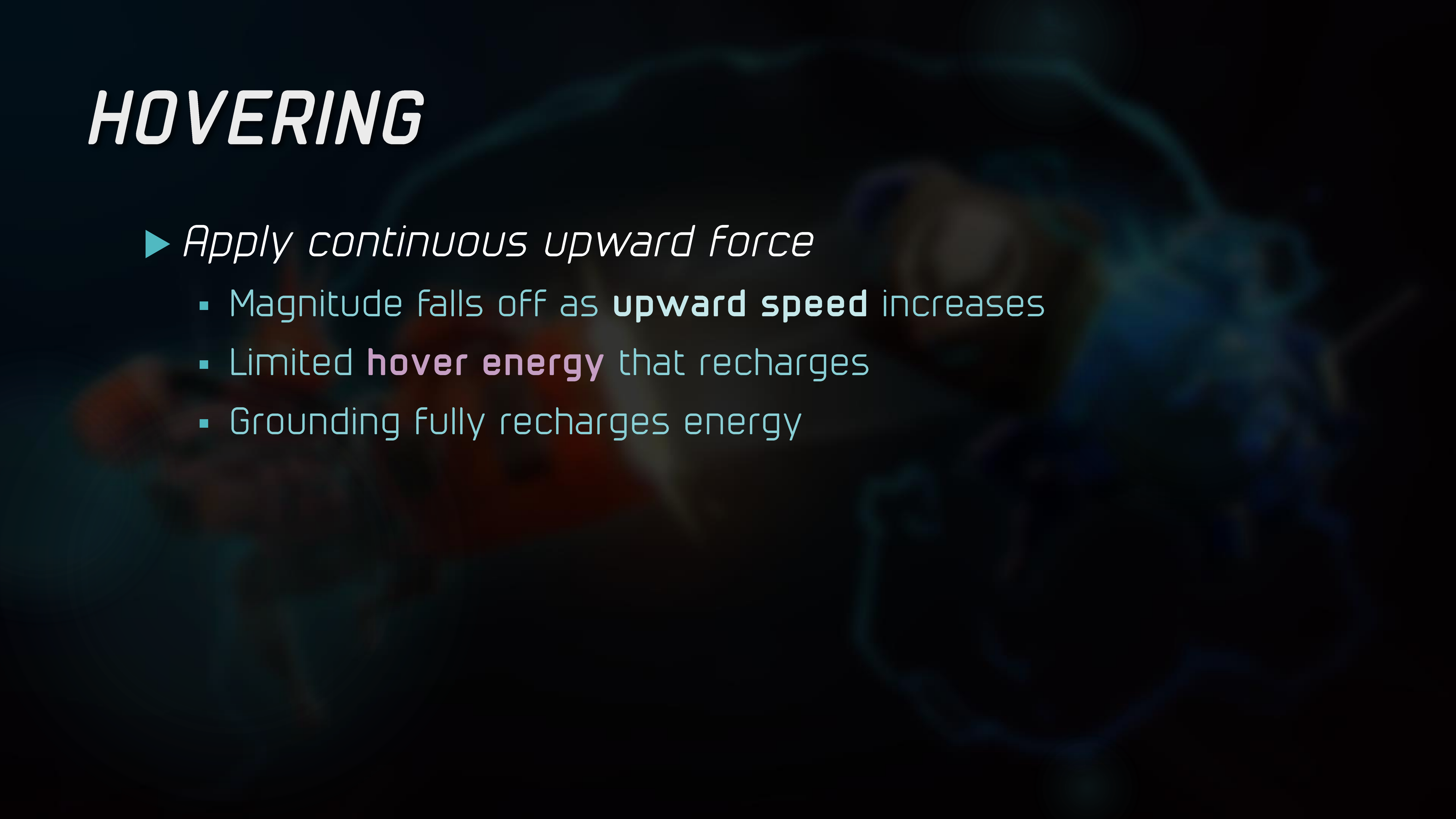
# CROUCHING

*no gameplay effect, just feels good!*

# MECHANICS



- Movement
- Braking
- Jumping
- Wall jumping
- Crouching
- **Hovering**
- Dashing
- Stomping
- Grabbing, Shooting



## *HOVERING*

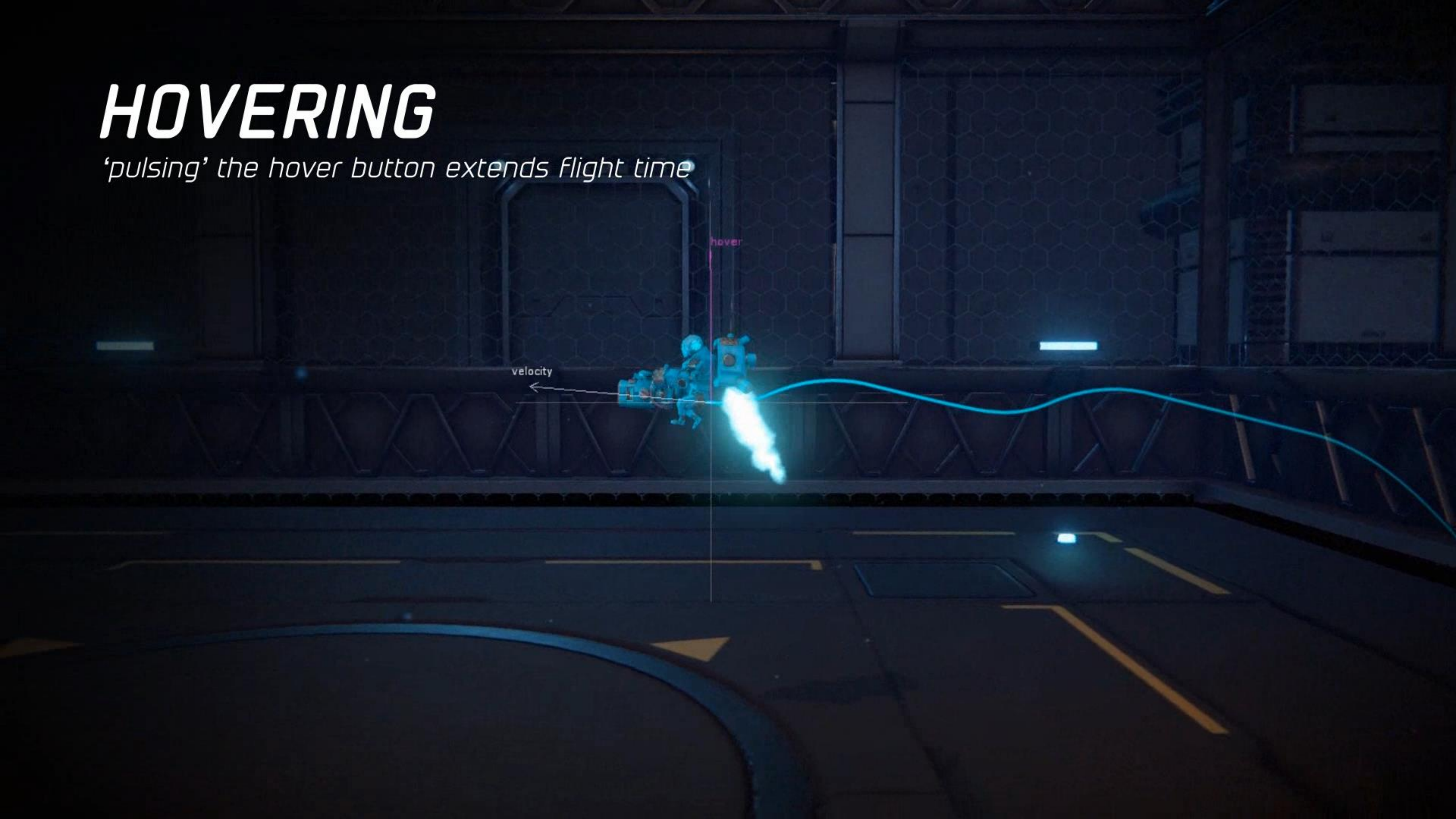
- *Apply continuous upward force*
  - Magnitude falls off as **upward speed** increases
  - Limited **hover energy** that recharges
  - Grounding fully recharges energy

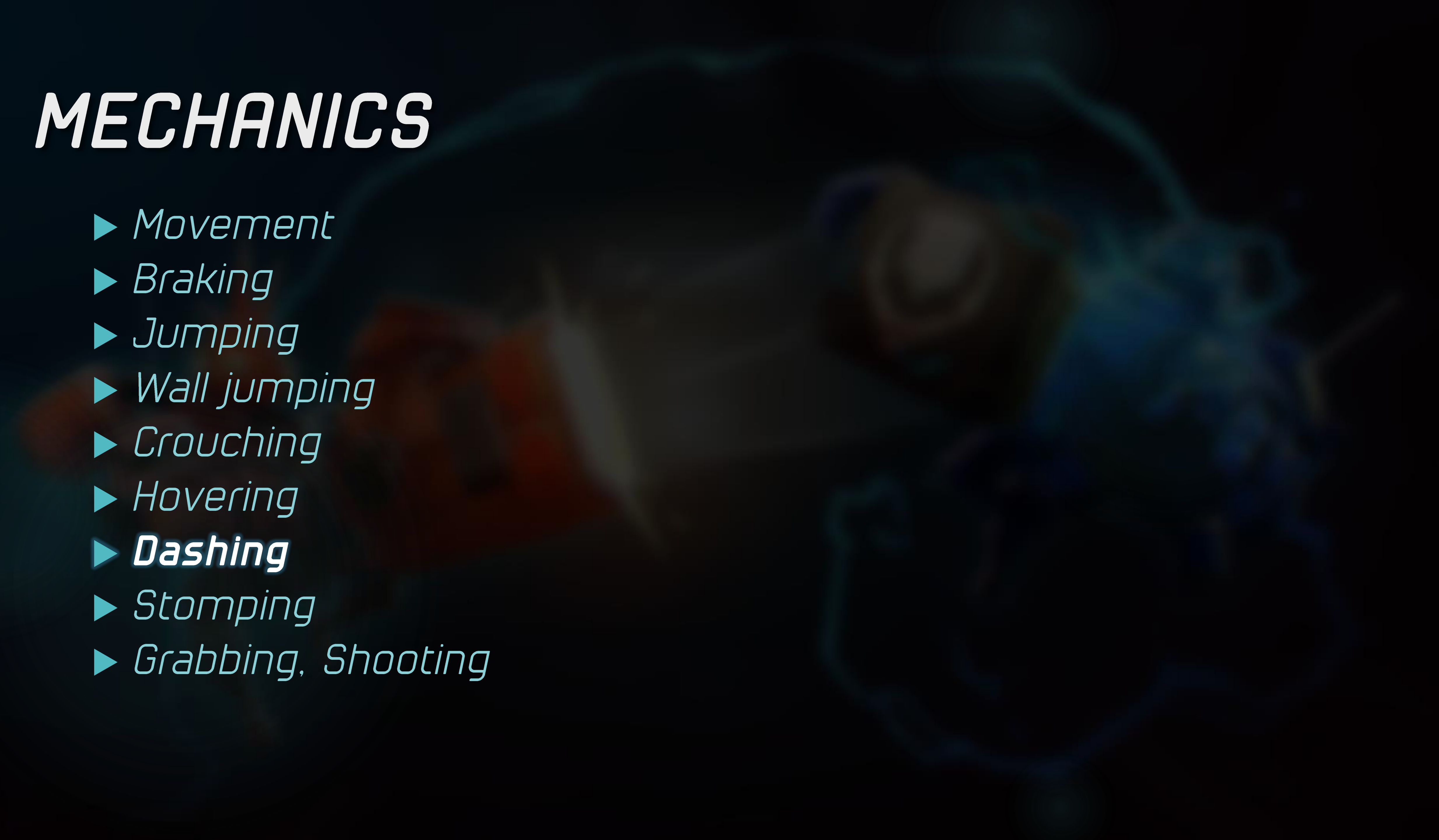
## HOVERING

```
// Apply hover force according to upward speed.
upSpeed = Vector3.Dot(Body.velocity, up);
scale = UpwardSpeedFalloff.Evaluate(upSpeed);
force = up * HoverForceMax * scale;
Body.AddForce(force);
```

## HOVERING

*'pulsing' the hover button extends flight time*





## MECHANICS

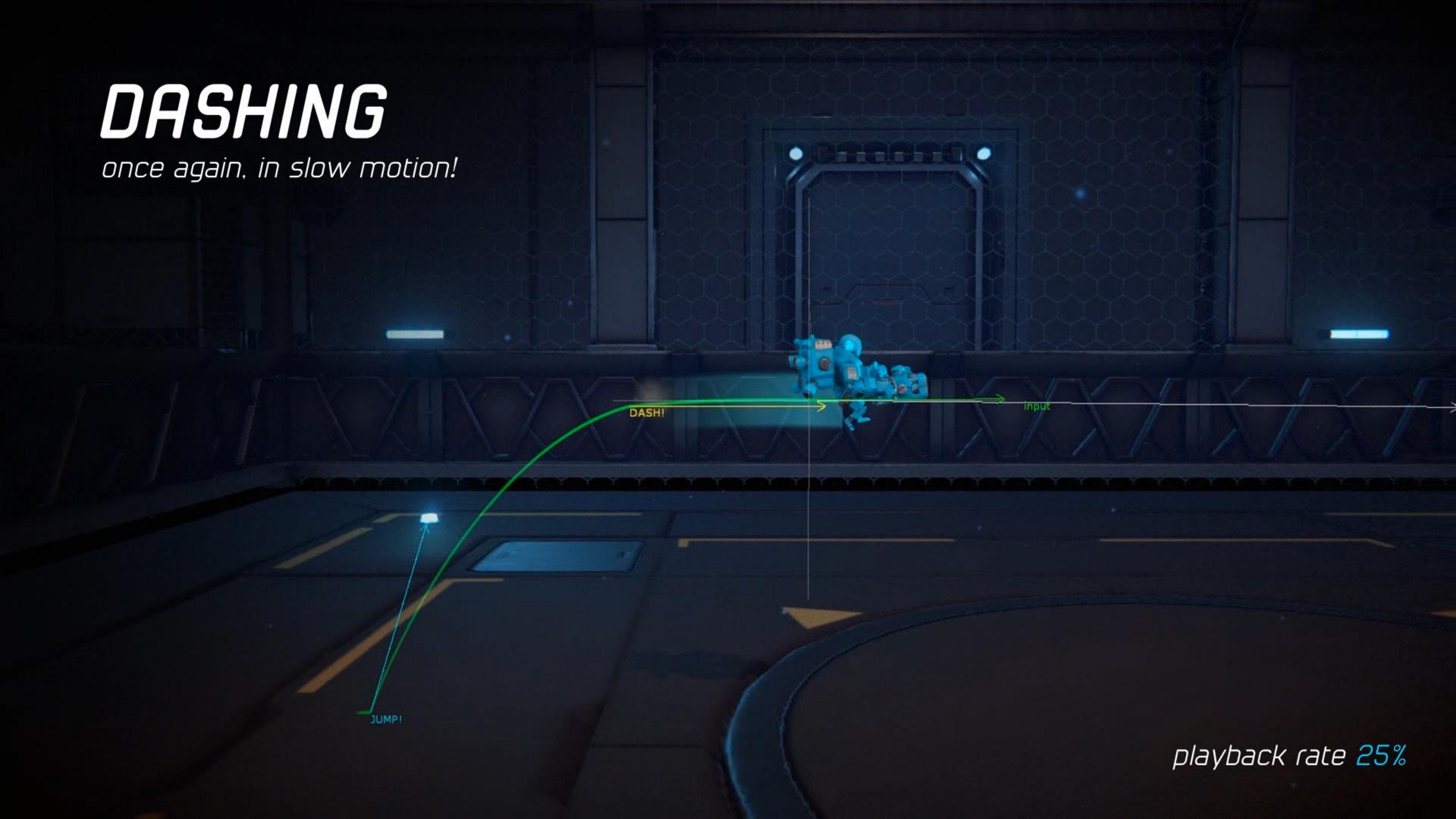
- Movement
- Braking
- Jumping
- Wall jumping
- Crouching
- Hovering
- **Dashing**
- Stomping
- Grabbing, Shooting

## DASHING

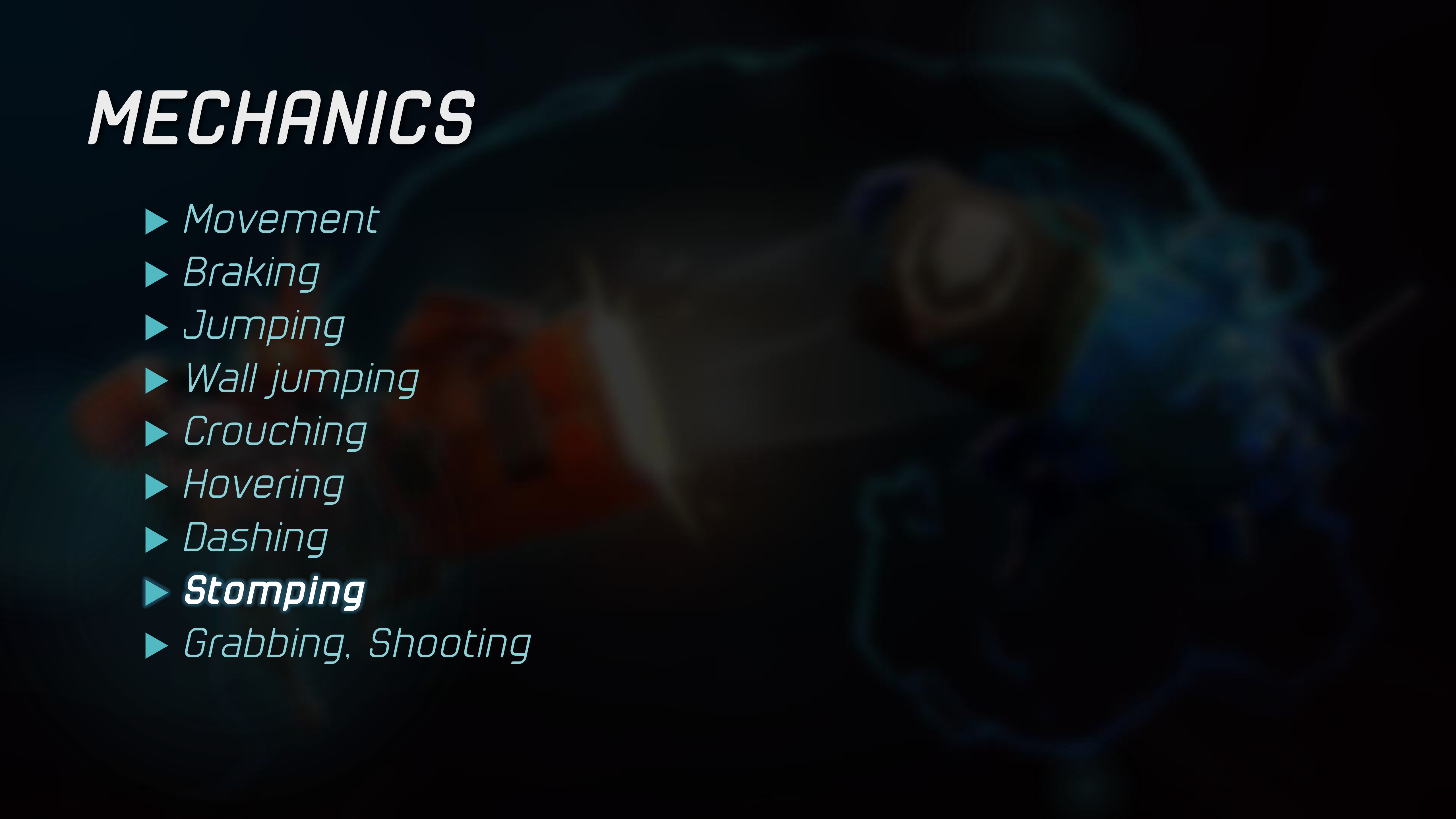
- *Increase* ***velocity****, disable* ***brakes*** *to* ***dash***
  - Very short duration
  - Cooldown between successive dashes
  - Restrict to cardinal directions
  - No up-dash (could stay aloft forever)

## DASHING

*once again, in slow motion!*



playback rate 25%



## MECHANICS

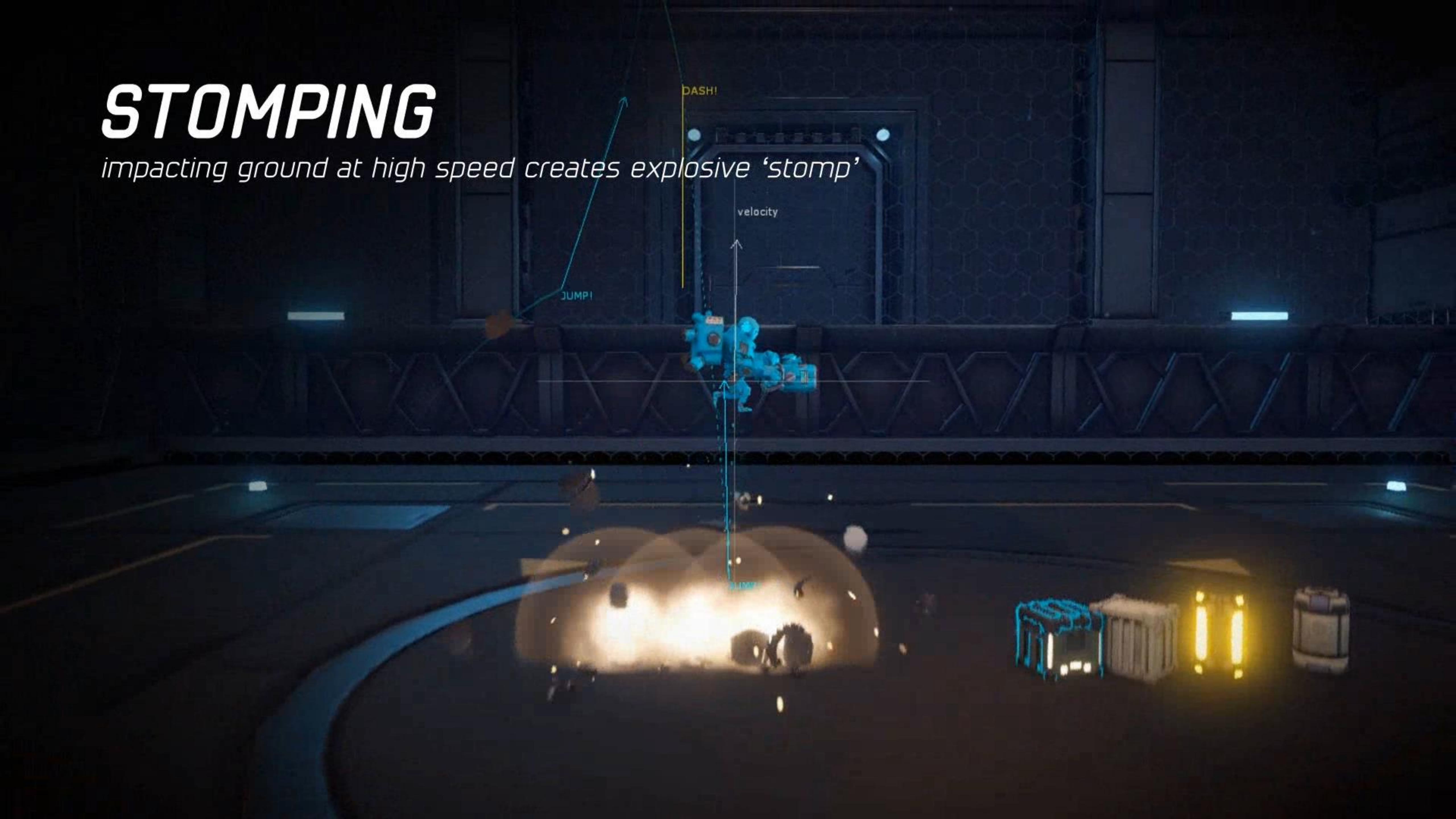
- Movement
- Braking
- Jumping
- Wall jumping
- Crouching
- Hovering
- Dashing
- **Stomping**
- Grabbing, Shooting

## STOMPING

- *When landing on ground,*
  - Check vertical **velocity**
  - If traveling fast enough, spawn **explosive** effect!
  - If lateral speed is low, also **jump**

## STOMPING

*impacting ground at high speed creates explosive 'stomp'*

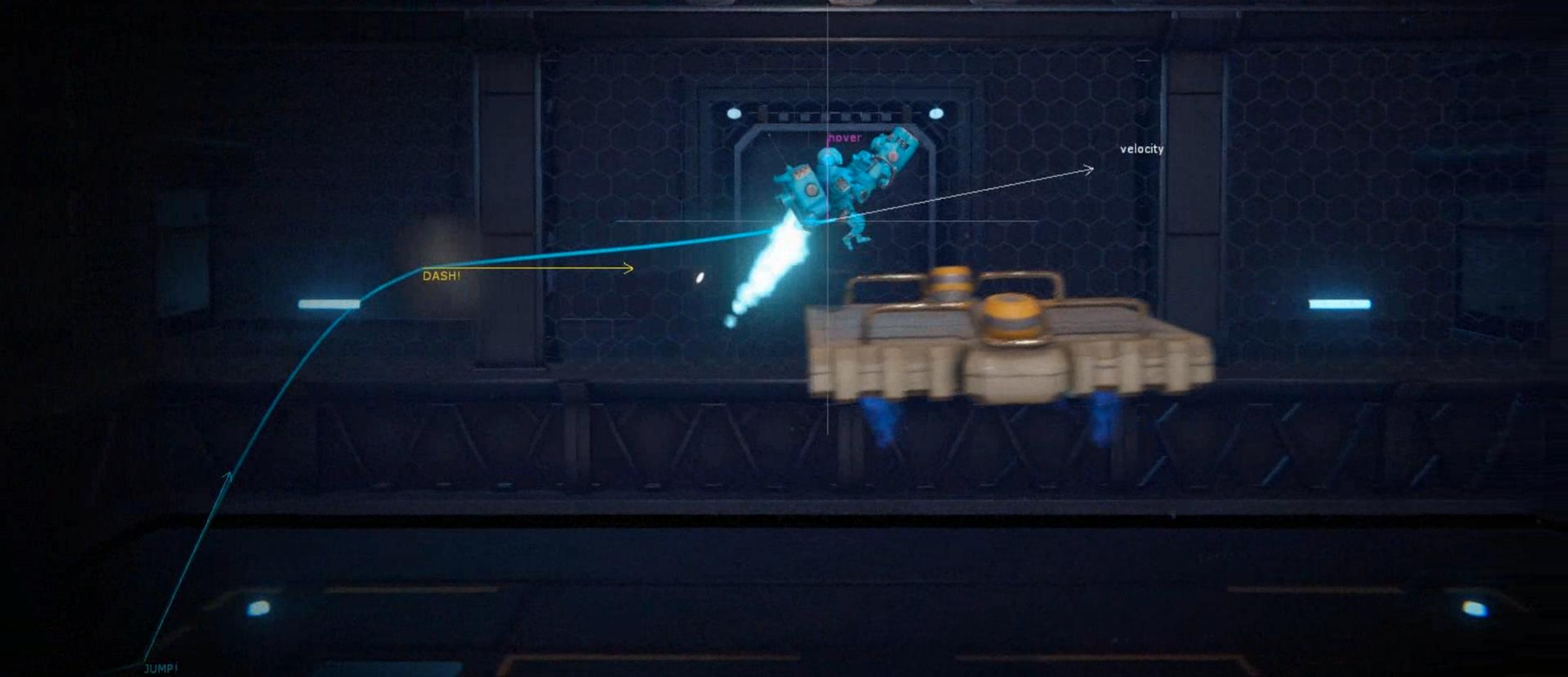


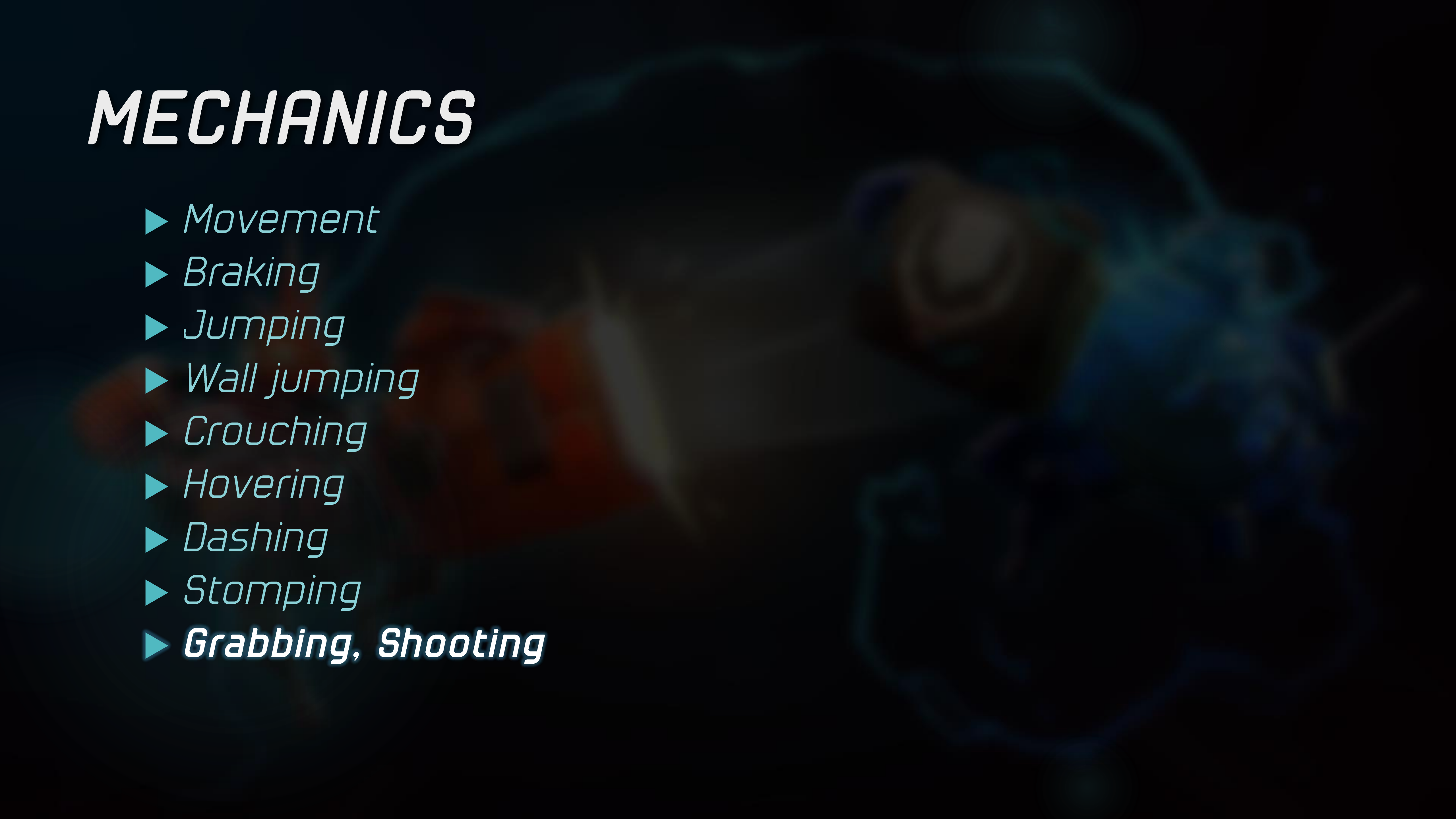
## AERIAL MANEUVERS

- **Combine** *to maximize hang-time!*
  - Jump, double jump, wall jump
  - Hover
  - Dash

# AERIAL MANEUVERS

combining jump, double-jump, wall-jump, hover and dash





## MECHANICS

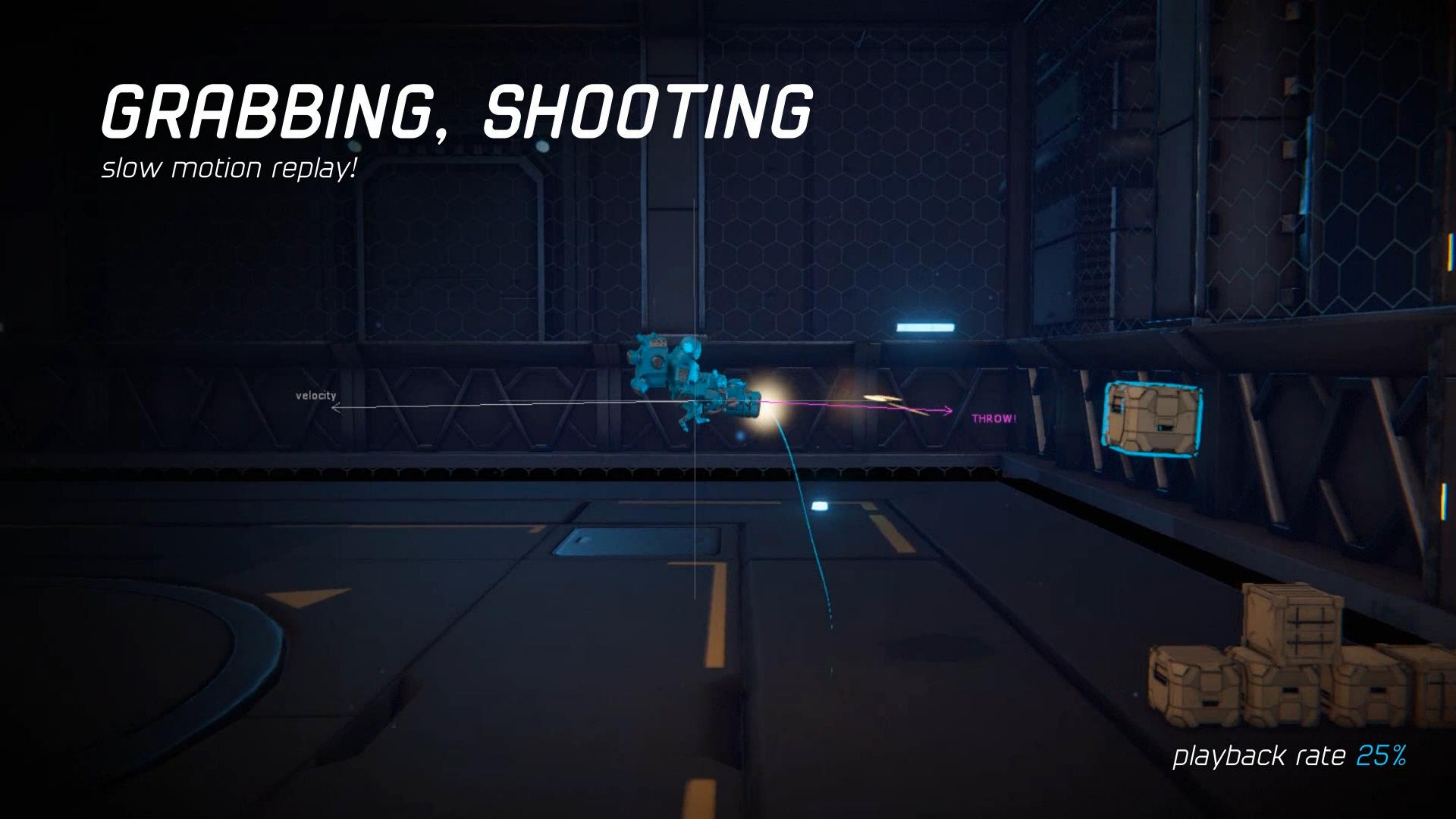
- Movement
- Braking
- Jumping
- Wall jumping
- Crouching
- Hovering
- Dashing
- Stomping
- **Grabbing, Shooting**

## GRABBING, SHOOTING

- *GRABBING*
  - Detect nearby objects, check LOS
  - Decide on a current **grab candidate**
  - Apply forces (using PID control) to attract object
  - Once **snapped** to gun, switch to kinematic
- *SHOOTING*
  - Just unsnap and apply a large **velocity**!
  - Add a **recoil force** to player

## GRABBING, SHOOTING

slow motion replay!



playback rate 25%

# PUTTING IT ALL TOGETHER

- Movement
- Braking
- Jumping
- Wall Jumping
- Crouching
- Hovering
- Dashing
- Stomping
- Grabbing, Shooting

$\Rightarrow$

**FLUID**

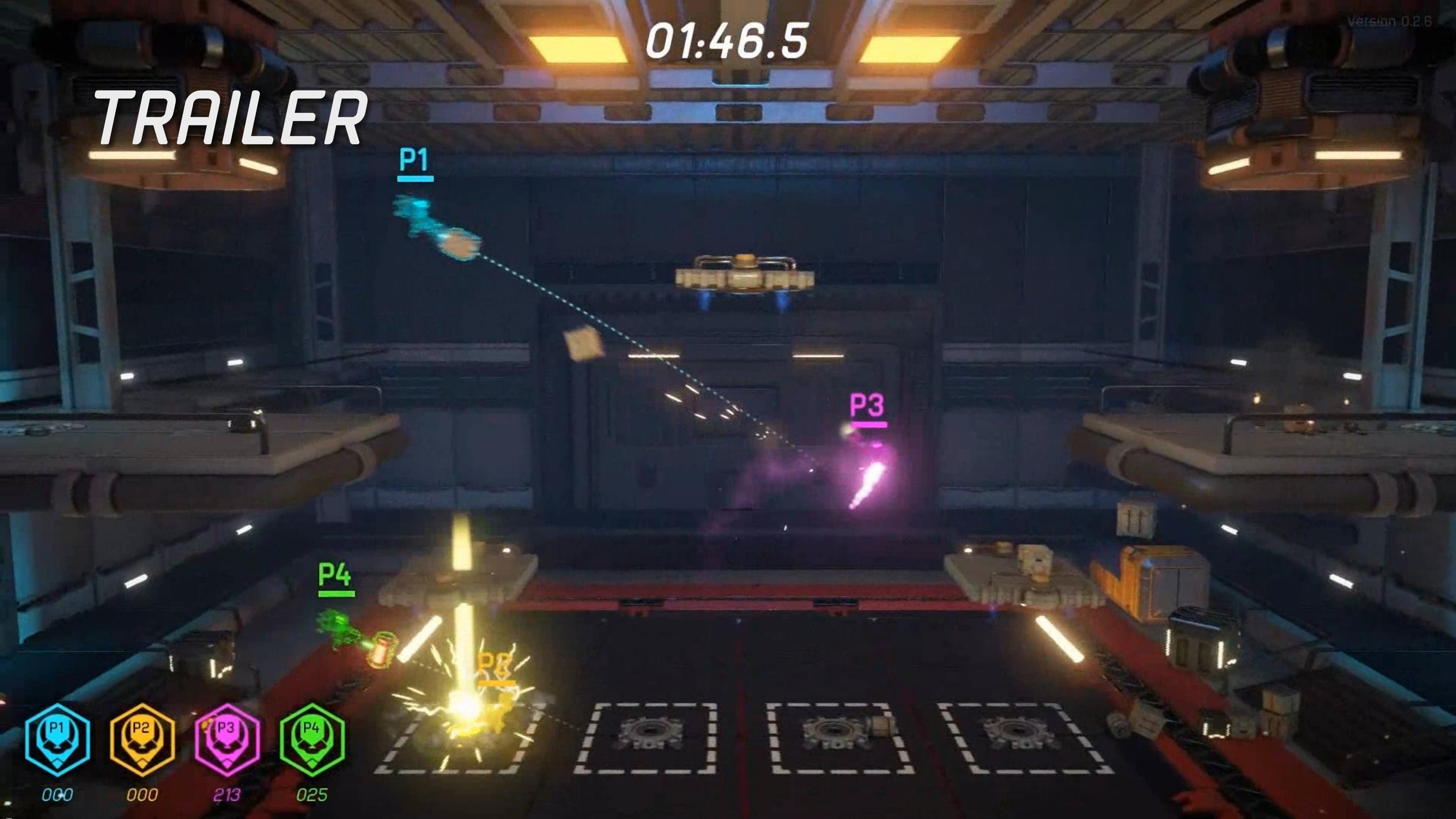
**CONTROL**

(hopefully)

## HAPPY ACCIDENTS

- *Some 'emergent' mechanics*
  - ***Blocking*** (using grabbed objects as shields)
  - ***Rocket jumping*** (firing objects down to boost up)
  - ***Bashing*** (sprinting + dashing into enemies)
  - ***Stomping*** and ***bouncing*** (latter was a bug!)

## TRAILER



# THANKS!

Come hang out at **Bean Bag End!**

- grabitygame.com
- @teamninjathumbs
- any questions?