Unity Game Engine

Introduction to Unity – Character Jump

Player Character Jump

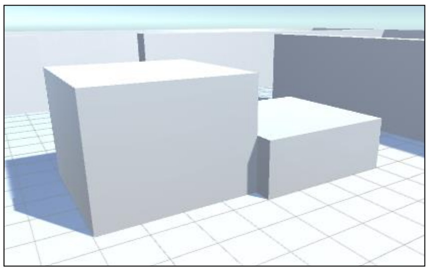
- Character jump is a pretty common movement in video games.
- Let's handle both jumping and falling.
  - Apply a velocity pulling the character (e.g player) down at all times.
  - Occasionally an upward jolt will be applied when the character jumps.

Animation for Character Jump

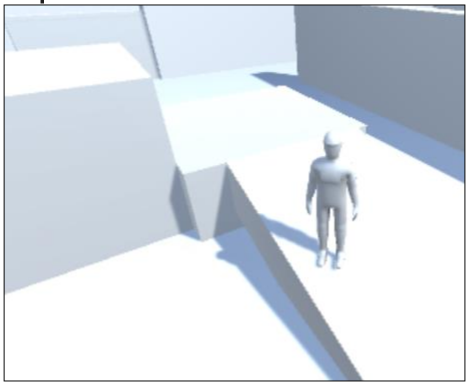
- Proper animations should be played for jumping.
  - Assume that the character has animations for idle, walk, and <u>jump</u>.
  - Idle or walk animation should be played when the character is on the ground.
  - <u>Jump animation should be played when the character is not on the ground.</u>

Jumping Environment

- Assume that we have a few raised platforms in the scene for the character to jump on or fall off of.



Jumping Environment (2)



- Have a ramp with slope.
  - Add a cube and set its transform values to Position -1.5, 1.5, 5 Rotation 0, 0, -25 Scale 1, 4, 4.

Player Character Moves

- Move on the ground
  - Changes on the x-z plane or x-z terrain.
- Move vertically
  - Move along y-axis
  - Move on un-even terrain
  - Jump
  - Flying
  - etc

Vertical Move for Jump

- Let’s make the character jump upon “**Space**” bar keyevent.
- Create a new script **Jump.cs** and attach it to the character.

Jump.cs (General Idea)

- The code will check if the character is on the ground. The vertical location will be adjusted differently depending on it.
  - If the character is on the ground, then the vertical speed should be reset to minFall to apply a velocity pulling the character down at all times.
  - If the character is not on the ground, then the vertical speed should be constantly reduced by gravity that acts a downward acceleration, resulting in a realistic falling movement.

Jump.cs

```
...
private CharacterController _charController;
public float jumpSpeed = 15.0f;
public float gravity = -9.8f;
public float terminalVelocity = -10.0f;
public float minFall = -1.5f;
private float _vertSpeed;
...
void Start() {
   _charController = GetComponent<CharacterController>();

   // Initialize the vertical speed to the minimum falling speed
   _vertSpeed = minFall;
  ...
}
```

Jump.cs

```
void Update() {
  ...
  // check if the controller is on the ground.
  if (_charController.isGrounded) {
    // React to the Jump button while on the ground.
    if (Input.GetButtonDown("Jump")) { // if spacebar pressed
      _vertSpeed = jumpSpeed;
    }
    else { // on the ground and jump is not initiated
      _vertSpeed = minFall;
    }
  }
```

`isGrounded` property of `CharacterController` indicates whether the bottom of the character controller collided with anything.

Jump.cs

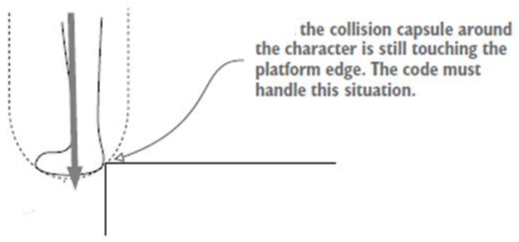
```
 else {  // If not on the ground,
    // then apply gravity until terminal velocity is reached.
   _vertSpeed +=  gravity * 5 * Time.deltaTime;
   if (_vertSpeed < terminalVelocity) {
     _vertSpeed = terminalVelocity;
   }
 }

// Set the vertical position
 movement.y = _vertSpeed;
 movement *= Time.deltaTime;
 _charController.Move(movement);

} // update
```

Ground Detection

- **`CharacterController.isGrounded`** property indicates whether the bottom of the character controller collided with anything.
- Using this Jump script version, the character will seem to float in the air while stepping off ledges. Similarly, if the character stands on a slope, the given ground detection template code will cause problematic behavior.

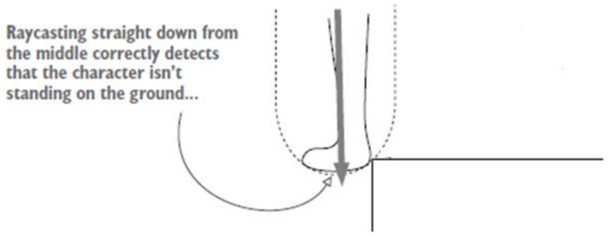
Ground Detection (2)

- That’s because the collision mesh (capsule) will still be in contact with the ground when the player steps off the edge of the platform.


the collision capsule around the character is still touching the platform edge. The code must handle this situation.

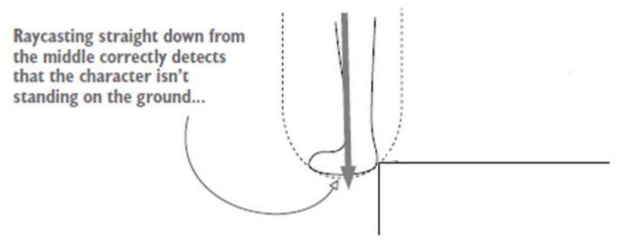
Ground Detection (3)

- Instead of using the collision mesh (capsule), let’s use raycasting to detect the ground.


Raycasting straight down from the middle correctly detects that the character isn’t standing on the ground...

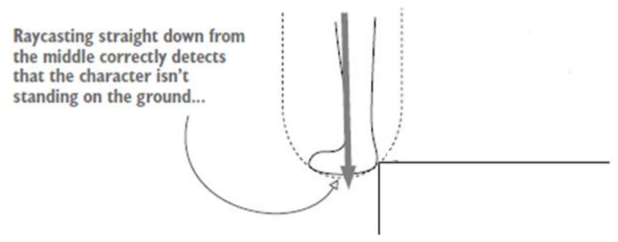
Ground Detection using Ray Casting

- Cast a ray straight down from the player's position.
  - If it registers a hit just below the character's feet, that means the player is standing on the ground.


Raycasting straight down from the middle correctly detects that the character isn't standing on the ground...

Ground Detection using Ray Casting

- When the raycast doesn't detect ground below the character but the character controller is colliding with the ground, the code should make the character slide off the ledge. (Special Case)


Raycasting straight down from the middle correctly detects that the character isn't standing on the ground...

Jump.cs (Modified)

```
...
private ControllerColliderHit _contact;
…
void Update() {
  …
  // raycast down to address steep slopes and dropoff edge
  bool hitGround = false;
  RaycastHit hit;

  if ( _vertSpeed < 0 &&
       Physics.Raycast(transform.position, Vector3.down, out hit))  {

    float check = (_charController.height + charController.radius)/1.9f;

    // to be sure check slightly beyond bottom of capsule
    hitGround = hit.distance <= check;
  }
```

1

To avoid using CharacterController.isGrounded, check if the ray hits the ground.

Modified Jump.cs

```
 if (hitGround) { // instead of using CharacterController.isGrounded
    if (Input.GetButtonDown("Jump")) {
      _vertSpeed = jumpSpeed;
    }
    else { // on the ground and jump is not initiated
      _vertSpeed = minFall;
    }
  }

 else { // If not on the ground,
    // apply gravity until terminal velocity is reached.
     _vertSpeed += gravity * 5 * Time.deltaTime;
     if (_vertSpeed < terminalVelocity) {
      _vertSpeed = terminalVelocity;
     }

     // special-case here
```

Modified Jump.cs

```
    if (_charController.isGrounded) { // special case – sliding off the ledge
        if (Vector3.Dot(movement, _contact.normal) < 0) {
          movement = _contact.normal * moveSpeed;
        }
        else {
          movement += _contact.normal * moveSpeed;
        }
    }
  } // else – if not on the ground

  movement.y = _vertSpeed;
  movement *= Time.deltaTime;
  _charController.Move(movement);
} // end update

// store collision data to use in Update
void OnControllerColliderHit(ControllerColliderHit hit) {
  _contact = hit;
}
```

2

3

Calculating `check` distance

- How the `check` distance is calculated?

1

```
check = (_charController.height + charController.radius)/1.9f;
```

  - First take the height of the character controller (which is the height without the rounded ends) and then add the rounded ends.
  - Divide this value in half because the ray was cast from the middle of the character (that is, already halfway down) to get the distance to the bottom of the character.
    - But we really want to check a little beyond the bottom of the character to account for tiny inaccuracies in the raycasting, so divide by 1.9 instead of 2 to get a distance that's slightly too far.

Ground Check with Raycast

- Raycasting - if the distance of the hit (`hit.distance`) is at the distance of the character's feet (`check`), then the character is standing on the ground, so set `hitGround` to true.

hitGround = hit.distance <= check;

1

Sliding Off

2

- The collision data (**`_contact)`** includes a `normal` property that tells us <u>the direction to move away from the point of collision</u>.
- The movement vector's facing relative to the point of collision can be determined using the dot product.
  Vector3.Dot(movement, _contact.normal)
- Vector3.Dot(movement, _contact.normal) < 0
  - Movement is opposite of the normal direction
  - Then, follow the normal direction
- Vector3.Dot(movement, _contact.normal) >= 0
  - Movement is aligned with the normal direction
  - Then, keep the movement momentum.

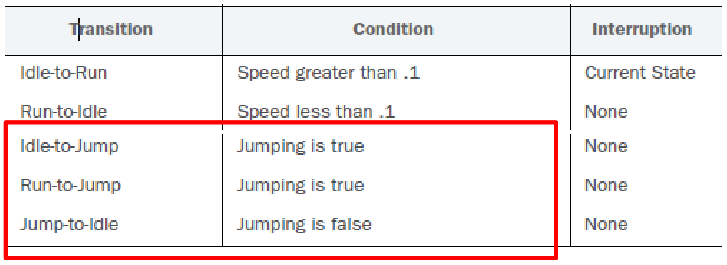
Collision Data

3

- Collisions with the character controller are reported through a callback function called `OnControllerColliderHit()`.
- In order use the collision data anywhere else in the script, the method stores the collision data in `_contact`.

Operating Animator

- Modify Jump.cs script to play proper animations based on the movements.
- In this example, “Speed” and “Jumping” control parameters are used to change animation states among “Idle”, “Run” and “Jump”. Transition to Jump has precedence.



Transition	Condition	Interruption
Idle-to-Run	Speed greater than .1	Current State
Run-to-Idle	Speed less than .1	None
Idle-to-Jump	Jumping is true	None
Run-to-Jump	Jumping is true	None
Jump-to-Idle	Jumping is false	None

Jump.cs (Modified)

```
...
private Animator _animator;
...
// Use this for initialization
void Start() {
  ...
  _animator = GetComponent<Animator>();
 …
}
```

Jump.cs (Modified)

```
…
 void Update() {
 …
   _animator.SetFloat("Speed", movement.sqrMagnitude);

   // raycast down to address steep slopes and dropoff edge
   bool hitGround = false;
   RaycastHit hit;

   if ( _vertSpeed < 0 &&
        Physics.Raycast(transform.position, Vector3.down, out hit))  {

     float check = (_charController.height + charController.radius)/1.9f;

     // to be sure check slightly beyond bottom of capsule
     hitGround = hit.distance <= check;
   }
```

Jump.cs (Modified)

```
    if (hitGround) {
      if (Input.GetButtonDown("Jump")) {
        _vertSpeed = jumpSpeed;
      }
      else {
        _vertSpeed = minFall;
        _animator.SetBool("Jumping", false);
    }
    else { // not on the ground
      _vertSpeed += gravity * 5 * Time.deltaTime;
      if (_vertSpeed < terminalVelocity) {
        _vertSpeed = terminalVelocity;
      }
      if (_contact != null ) {         // condition is not true only at level start
            _animator.SetBool("Jumping", true);
      }
```

Jump.cs (Modified)

```
      if (_charController.isGrounded) {
          if (Vector3.Dot(movement, _contact.normal) < 0) {
            movement = _contact.normal * moveSpeed;
          }
          else {
            movement += _contact.normal * moveSpeed;
          }
      }
    }
    movement.y = _vertSpeed;
    movement *= Time.deltaTime;
    _charController.Move(movement);
  } // end update

  // store collision to use in Update
  void OnControllerColliderHit(ControllerColliderHit hit) {
    _contact = hit;
  }
```

Operating Animator

- When the character is not on the ground, Jumping is set to true using _animator.SetBool("Jumping", true);
  - However, the condition (_contact != null) is there before setting the Jumping Boolean. It is always true except the level start.
  - That condition prevents the animator from playing the jump animation right from the start.
  - Even though the character is technically falling for a split second, there won't be any collision data until the character touches the ground for the first time.

Resources

- Jump.cs
  - This code handles vertical moves only. So the character should be handled in conjunction with a script moving the character forward and rotating.