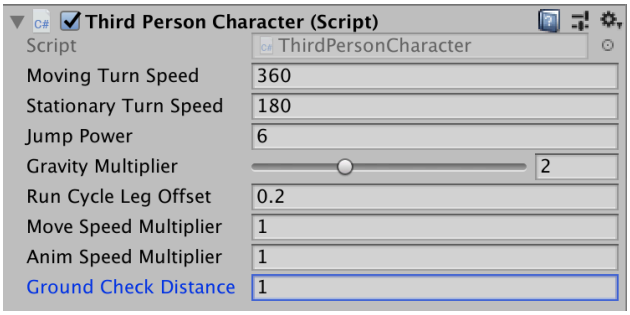
Quest 10 - Steps

1) Setup

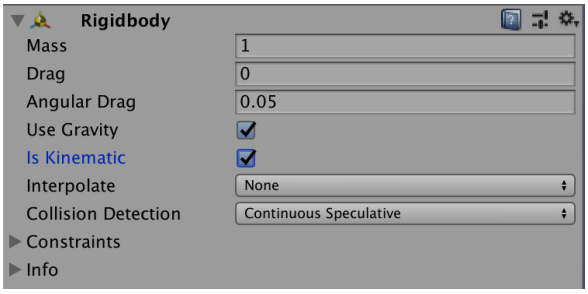
Continue from the camera and player mechanics established in Quest 9.

2) Some Bug Fixes

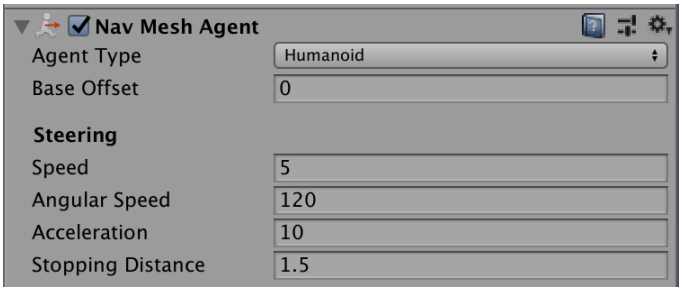
You may have also noticed that in some terrain layouts, the third person character appears to float. Increasing the Ground Check Distance can help if you experience this bug.



You may also have a bug where you can only intermittently select characters. This comes from a variety of factors where the physics state does not match the visual state of what we think we're clicking. Setting the Rigidbody to Kinematic (because NavMeshAgent will compete with Physics) and collision detection to Continuous Speculative can help.



Finally, because a Kinematic Rigidbody will defer to NavMeshAgent, we need to increase the agent's speed so it matches our prior physics-based movement. The increased Stopping Distance will help our characters better move in groups.



3) Multiple Unit Selection Box

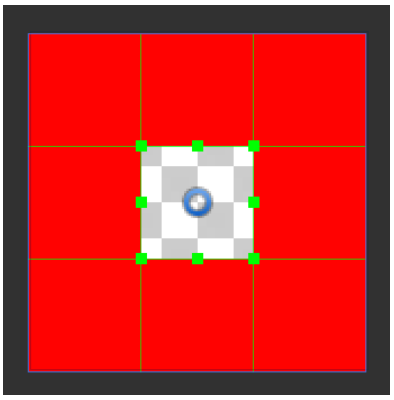
Drag outline.png to /Assets/Resources/Textures/

Change it's Import Settings to Texture Type = Sprite (2D and UI), Filter Mode = Point, and Compression = None.

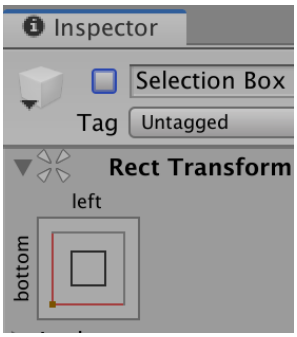
Open the Sprite Editor. (Depending on your Unity project settings, it may ask you to install the "2D Sprite" package using Window->Package Manager.

Drag the green handles inward from the edges to create a 9-slice grid of the sprite. This is used to specify which sections stretch when making a resizable UI.

Apply any changes as requested.



Create an UI->Image and name it "Selection Box". Assign the outline.png as the Source Image and Image Type = Sliced, Fill Center = Unchecked. Ensure that the Anchoring is set for Bottom-Left and uncheck to deactivate Selection Box for now.



In RTSGameController.cs, add outlets to reference the UI Canvas and Selection Box. mouseDragThreshold will configure how far the mouse has to move before it's considered a drag, and mouseClickStart will keep track of where the drag started.

```
// Outlets
public Canvas uiCanvas;
public RectTransform selectionBox;

// Configuration
float mouseDragThreshold = 3f;

// State Tracking
public GameObject currentSelection;
public Vector2 mouseClickStart;
```

Within Update, RENAME your GetMouseDown(0) function from Q9 to GetMouseUp(0). Add a line to hide the selection box when the mouse button is released.

```
// Selection Command
if(Input.GetMouseButtonUp(0)) {
    // Hide selection box
    selectionBox.gameObject.SetActive(false);
```

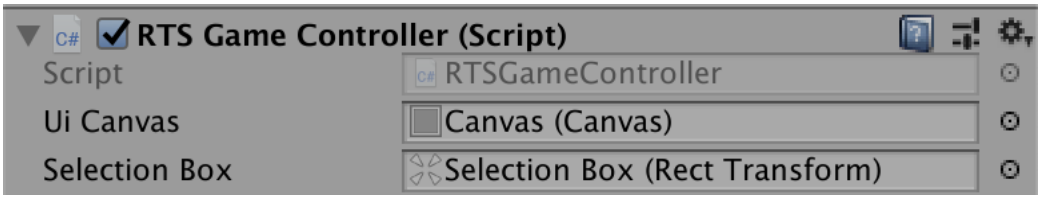
Also inside Update, create a new conditional for the GetMouseDown(0) scenario to keep track of where the drag started.

```
// Start Drag
if(Input.GetMouseButtonDown(0)) {
    mouseClickStart = Input.mousePosition;
}
```

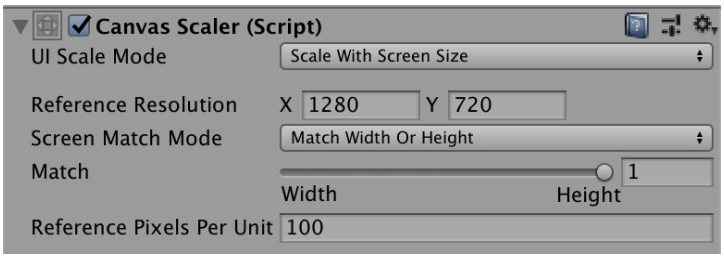
When the mouse button is held and dragged, we can use this movement to compute the position and size of a rectangle.

```
// Handle Drag
if(Input.GetMouseButton(0)) {
    Vector2 mousePosition = Input.mousePosition;
    if((mousePosition - mouseClickStart).magnitude > mouseDragThreshold) {
        selectionBox.gameObject.SetActive(true);

        // Compute midpoint
        selectionBox.anchoredPosition = Vector2.Lerp(mouseClickStart, mousePosition, 0.5f) / uiCanvas.scaleFactor;

        Vector2 box = new Vector2(Mathf.Abs(mouseClickStart.x - mousePosition.x) / uiCanvas.scaleFactor,
                                  Mathf.Abs(mouseClickStart.y - mousePosition.y) / uiCanvas.scaleFactor);
        selectionBox.sizeDelta = box;
    }
}
```

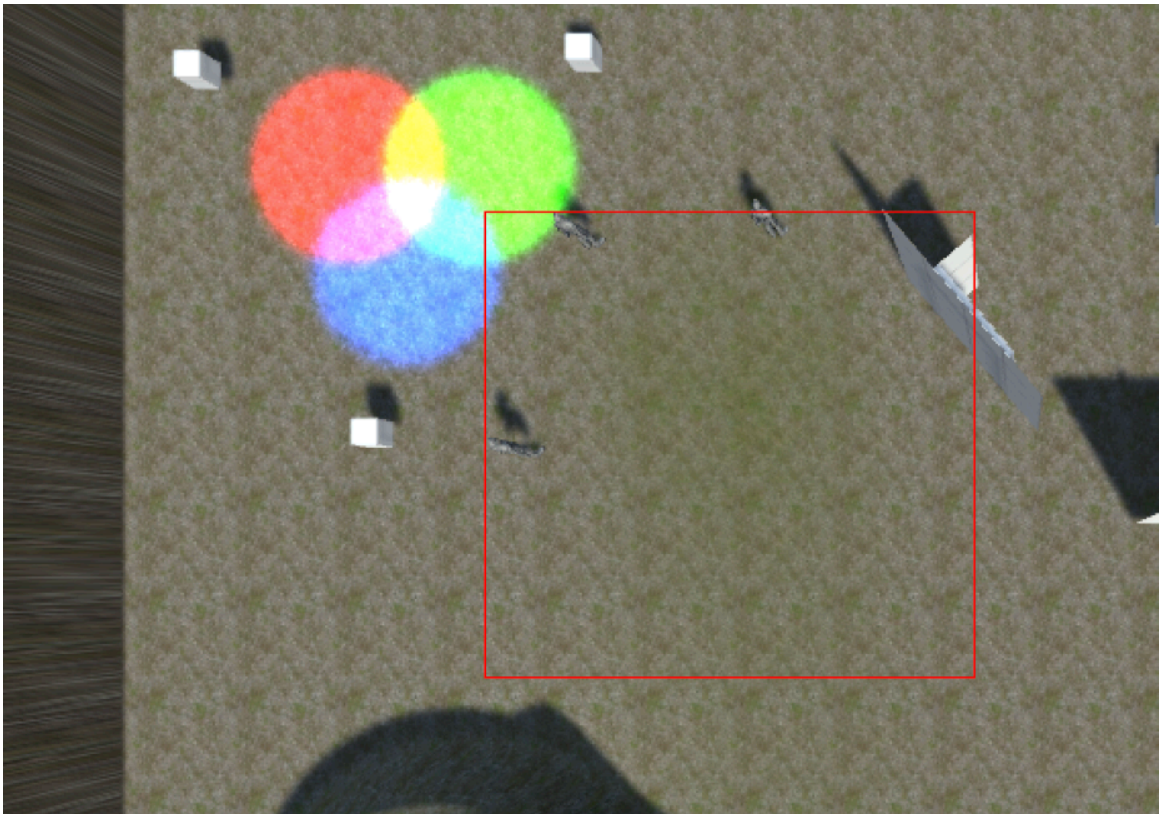
Fill in the blanks created by the new outlets.



Select the canvas and set up the scaler to provide a more consistent UI experience.



Playtest the game to ensure a selection box is appropriately shown, resized, and hidden when dragging and releasing the left-click.



4) Multiple Unit Selection Functionality

The old character selection code needs to be refactored to work with lists instead of single game objects.

```
// State Tracking
public List<GameObject> currentSelection;
public Vector2 mouseClickStart;

// Methods
void Start() {
    currentSelection = new List<GameObject>();
}
```

On mouse release, we raycast and any compatible objects found are added to the list, otherwise the list is emptied.

```
// Selection Command
if(Input.GetMouseButtonUp(0)) {
    currentSelection.Clear();

    // Hide selection box
    selectionBox.gameObject.SetActive(false);

    // Get all objects under the mouse cursor
    Ray selectionRaycast = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit[] hits = Physics.RaycastAll(selectionRaycast);

    // See if any are selectable
    foreach(RaycastHit hit in hits) {
        if(hit.collider.GetComponent<RTSCharacterController>()) {
            currentSelection.Add(hit.collider.gameObject);
        }
    }
}
```

On right-click, the code is almost exactly the same as last quest, except we perform that work for every item in the list.

```
// Interact Command
if(Input.GetMouseButtonDown(1) && currentSelection.Count > 0) {
    foreach(GameObject selection in currentSelection) {
        RTSCharacterController rtsCharacterController = selection.GetComponent<RTSCharacterController>();
        if(rtsCharacterController) {
            // Get all objects under the mouse cursor
            Ray selectionRaycast = Camera.main.ScreenPointToRay(Input.mousePosition);
            RaycastHit[] hits = Physics.RaycastAll(selectionRaycast);

            // Check for possible interactions
            foreach(RaycastHit hit in hits) {
                if(hit.collider.gameObject.layer == LayerMask.NameToLayer("Ground")) {
                    // Move player to destination
                    rtsCharacterController.SetDestination(hit.point);
                }
            }
        }
    }
}
```

Playtest your game and double-check that single character selection and movement still works as expected.

Currently, releasing the left-click performs a raycast at the point occupied by the mouse position. To do multi-unit selection, we will take a different perspective. Instead of casting to see if an object is hit, selectable objects will convert their world space positions to screen positions, and we will see if those screen positions are within the selection box

First lets re-organize the code because the amount of functionality is becoming unwieldy. The SelectUnderMouse() function encompasses our existing code for selecting a single character with a left-click.

```
void SelectUnderMouse() {
    // Get all objects under the mouse cursor
    Ray selectionRaycast = Camera.main.ScreenPointToRay(Input.mousePosition);
    RaycastHit[] hits = Physics.RaycastAll(selectionRaycast);

    // See if any are selectable
    foreach(RaycastHit hit in hits) {
        if(hit.collider.GetComponent<RTSCharacterController>()) {
            currentSelection.Add(hit.collider.gameObject);
        }
    }
}
```

Similarly, the SelectWithinBox() function encompasses the code for selecting all eligible objects within the selection box.

```
void SelectWithinBox() {
    // Expensive. In a real-game, you would maintain a list instead
    RTSCharacterController[] characterControllers = FindObjectsOfType<RTSCharacterController>();

    // Loop through each eligible object to see if it's in our selection box
    foreach(RTSCharacterController character in characterControllers) {
        // Translate its position to screen space
        Vector2 characterPosition = Camera.main.WorldToScreenPoint(character.transform.position);

        // Adjust screen space for any scaling
        characterPosition = characterPosition / uiCanvas.scaleFactor;

        // Normalize for our UI box's center pivot and bottom-left anchor
        Rect anchoredRect = new Rect(selectionBox.anchoredPosition.x - selectionBox.rect.width / 2f,
            selectionBox.anchoredPosition.y - selectionBox.rect.height / 2f,
            selectionBox.rect.width,
            selectionBox.rect.height);

        // Does the character fall inside the box?
        if(anchoredRect.Contains(characterPosition)) {
            currentSelection.Add(character.gameObject);
        }
    }
}
```

Finally, we revise our GetMouseButtonUp(0) conditional within the Update loop to use these functions.

```
// Selection Command
if(Input.GetMouseButtonUp(0)) {
    currentSelection.Clear();

    if(selectionBox.gameObject.activeInHierarchy) {
        // Mouse Drag = Get all eligible objects
        SelectWithinBox();
    }
    else {
        // Single Click
        SelectUnderMouse();
    }

    // Hide selection box
    selectionBox.gameObject.SetActive(false);
}
```

Playtest selecting multiple characters in a variety of formations, to ensure the selection area is actually working.

5) Selection Indicators

In RTSCharacterController, add these two functions to highlight and unhighlight characters based on their selection status.

```
public void Select() {
    GetComponentInChildren<Renderer>().material.color = Color.blue;
}

public void Deselect() {
    GetComponentInChildren<Renderer>().material.color = Color.white;
}
```

Back in RTSGameController, create a DeselectAll function to handle both clearing the character list and unhighlighting them.

```
void DeselectAll() {
    foreach(GameObject selection in currentSelection) {
        selection.SendMessage("Deselect", SendMessageOptions.DontRequireReceiver);
    }
    currentSelection.Clear();
}
```

In the Update loop's GetMouseButtonUp(0) conditional, replace currentSelection.Clear() with a reference to the DeselectAll function.

```
// Selection Command
if(Input.GetMouseButtonUp(0)) {
    DeselectAll();
```

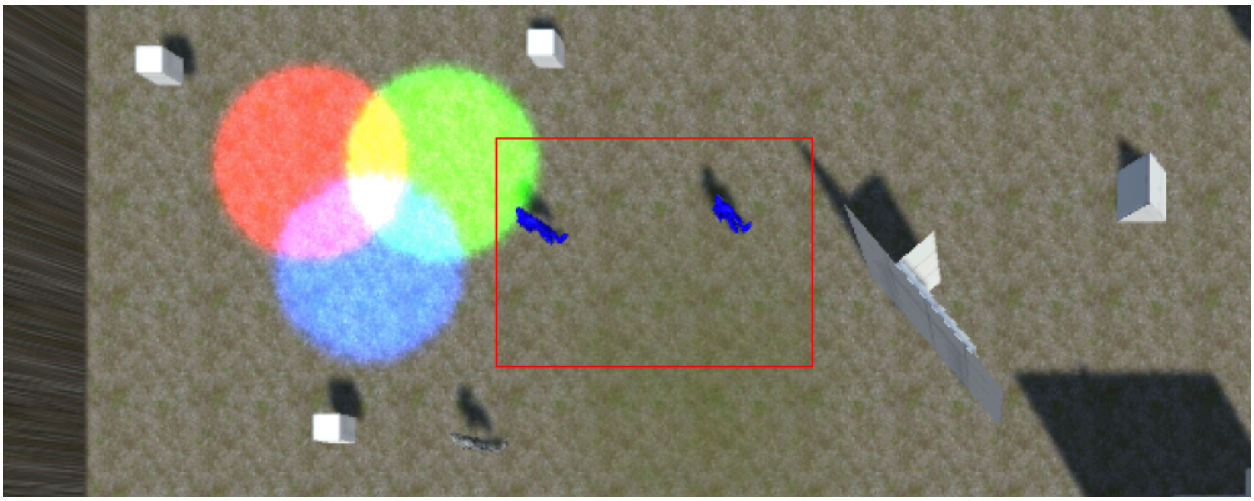
In the SelectUnderMouse function, add a line to trigger the Select command.

```
if(hit.collider.GetComponent<RTSCharacterController>()) {
    currentSelection.Add(hit.collider.gameObject);
    hit.collider.SendMessage("Select", SendMessageOptions.DontRequireReceiver);
}
```

In the SelectWithinBox function, add a similar line to trigger the Select command.

```
if(anchoredRect.Contains(characterPosition)) {
    currentSelection.Add(character.gameObject);
    character.gameObject.SendMessage("Select", SendMessageOptions.DontRequireReceiver);
}
```

Active characters are now highlighted.

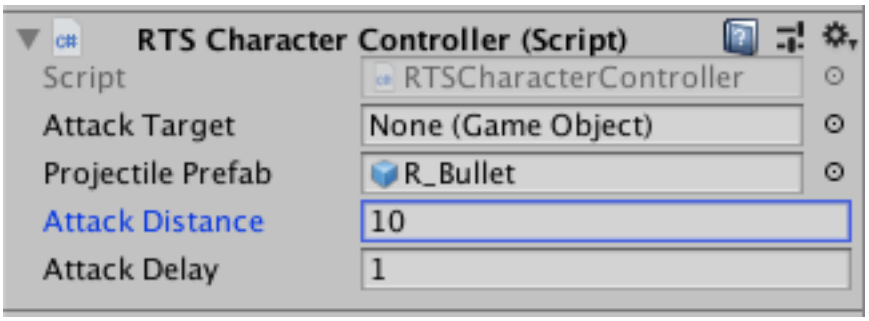


6) Player Attacks

For the attack functionality, RTSCharacterController needs outlets for its target and projectile prefab as well as variables that defining attack timing.

```
public class RTSCharacterController : MonoBehaviour
{
    // Outlets
    public GameObject attackTarget;
    public GameObject projectilePrefab;

    // Configuration
    public float attackDistance;
    public float attackDelay;

    // State Tracking
    float _timeSinceLastAttack;
```

In the Inspector for each character, use the following settings.



The SetTarget function will allow the game controller to pass targets to the character when the player right-clicks.

```
public void SetTarget(GameObject target) {
    attackTarget = target;
}
```

Within Update of RTSCharacterController, we will increment the attack timer counting up until it is time to attack. We set up the conditional that will handle when the character has a target.

```
void Update() {
    // Increment attack timer
    _timeSinceLastAttack += Time.deltaTime;

    // Prepare to get in range of target
    if(attackTarget) {

    }

}
```

Within that conditional, we tell the character to move toward its target and create a conditional to check if we are in attack range.

```
// Prepare to attack
SetDestination(attackTarget.transform.position);
if(Vector3.Distance(attackTarget.transform.position, transform.position) <= attackDistance ) {

}
```

Finally, inside of that conditional, if we are in attack range, we check if enough time has elapsed for us to perform an attack. If so, we reset the timer again and create a physics projectile we send towards the target.

```
// Perform attack
if(_timeSinceLastAttack >= attackDelay) {
    // Reset attack timer
    _timeSinceLastAttack = 0;

    // Prepare attack
    Vector3 projectileOrigin = transform.position + new Vector3(0, 2f, 0);
    Vector3 directionToTarget = (attackTarget.transform.position - projectileOrigin).normalized;

    // Shoot projectile
    GameObject projectile = Instantiate(
        projectilePrefab,
        projectileOrigin,
        Quaternion.LookRotation(directionToTarget)
    );
    projectile.transform.localScale = Vector3.one * 10f;
    projectile.GetComponent<Rigidbody>().AddForce(directionToTarget * 50f, ForceMode.Impulse);

    // Cleanup projectile
    Destroy(projectile, 10f);
}
```

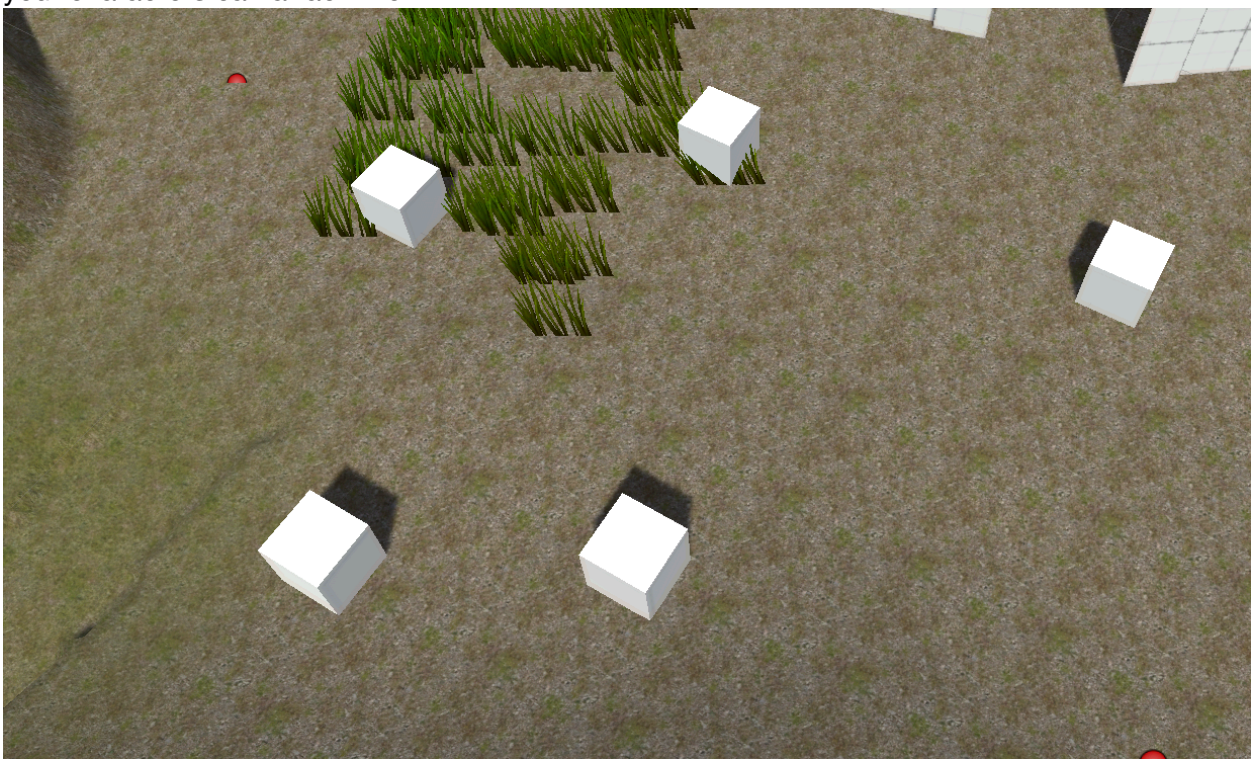
7) Triggering Attacks

In RTSGameController, we will make tweaks to the right-click functionality that will trigger an attack. The move command will still work if no targets are available.

```
// Get all objects under the mouse cursor
Ray selectionRay = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit[] hits = Physics.RaycastAll(selectionRay);
foreach(RaycastHit hit in hits) {
    if(hit.collider.gameObject.layer != LayerMask.NameToLayer("Ground")) {
        rtsCharacterController.SetTarget(hit.collider.gameObject);
        break;
    }

    if(hit.collider.gameObject.layer == LayerMask.NameToLayer("Ground")) {
        rtsCharacterController.SetTarget(null);
        rtsCharacterController.SetDestination(hit.point);
    }
}
```

8) Environment Targets

Create several physics-enabled objects spread around your level and playtest to make sure your characters can attack them.



Players can chase and attack any object with a collider, even enemies.