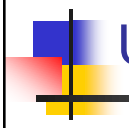


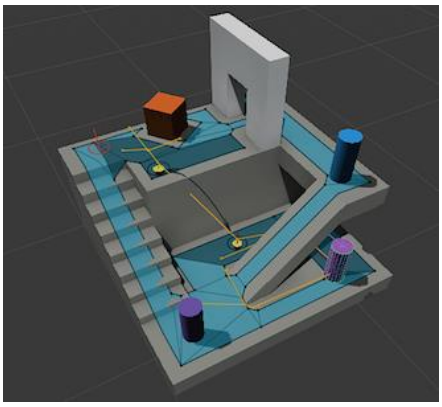
Unity Game Engine

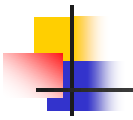
Introduction to Unity – Navigation and Pathfinding

https://unity3d.com/learn/tutorials/topics/navigation
https://docs.unity3d.com/Manual/Navigation.html

Navigation and Pathfinding

- Unity **navigation system** allows you to create characters that can intelligently move around the game world.





Unity Navigation System

- The Unity Navigation system consists of the following pieces:
  - **NavMesh**
  - **NavMesh Agent**
  - **NavMesh Obstacle**
  - **Off-Mesh Link**

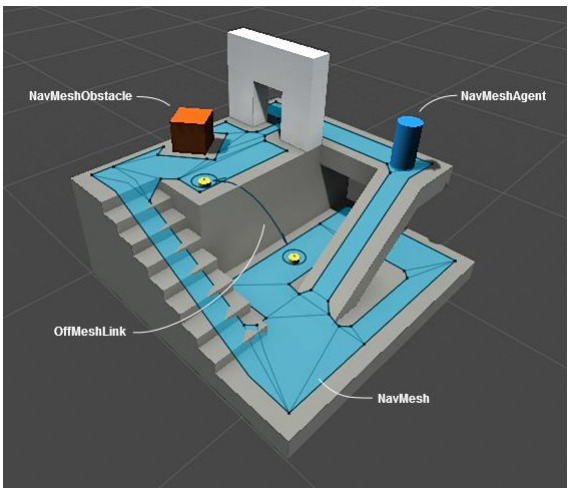
Unity Navigation System

- **NavMesh** (short for Navigation Mesh) is a data structure which describes the *walkable surfaces* of the game world. Created by NavMesh Baking.
- **NavMesh Agent** component help you to *create navigating characters* which avoid each other while moving towards their goal.

Unity Navigation System

- **NavMesh Obstacle** component allows you to *describe moving obstacles* the agents should avoid while navigating the world.
- **Off-Mesh Link** component allows you to incorporate *navigation shortcuts* which cannot be represented using a walkable surface. For example, jumping over a ditch or a fence, or opening a door before walking through it.

Unity Navigation System

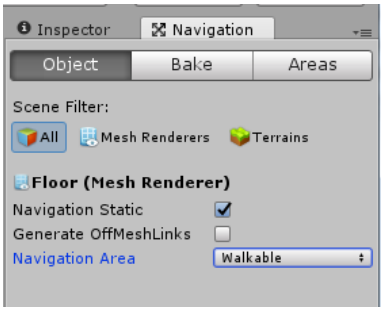


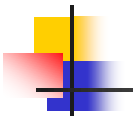
Unity Navigation System

- The Unity Navigation system consists of the following pieces:
  - **NavMesh**
  - **NavMesh Agent**
  - **NavMesh Obstacle**
  - **Off-Mesh Link**

NavMesh Baking

- **NavMesh Baking** - the process of creating a NavMesh from the level geometry.
- NavMesh generation is handled from the Navigation window via **Window > Navigation**





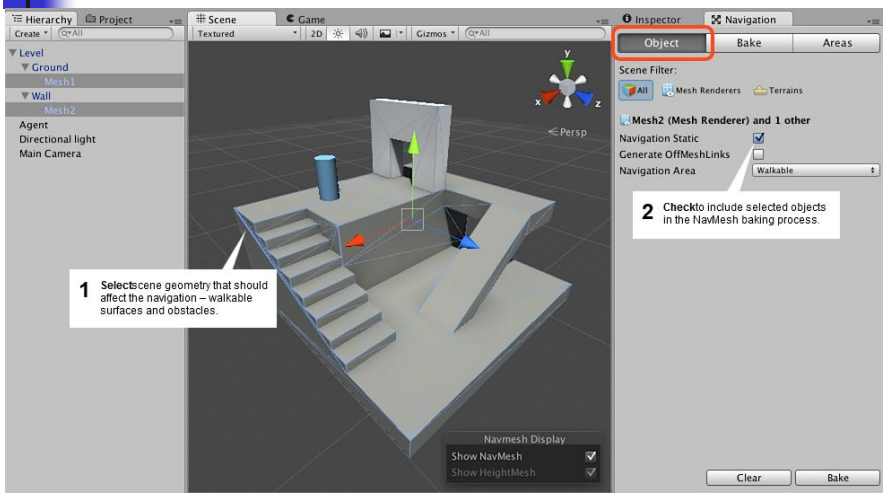
NavMesh Baking Steps

- Building a NavMesh for your scene can be done in 4 quick steps:
  - **1. Select** scene geometry.
  - **2. Check Navigation Static on**.
  - **3. Adjust the bake settings** to match your agent size.
  - **4. Click bake** to build the NavMesh.
- The resulting NavMesh will be shown in the scene as a blue overlay on the underlying level geometry.

NavMesh Baking – Steps (1,2)

- **1. Select** scene geometry that should affect the navigation – walkable surfaces and obstacles.
- **2. Check "Navigation Static" on** to include selected objects in the NavMesh baking process.

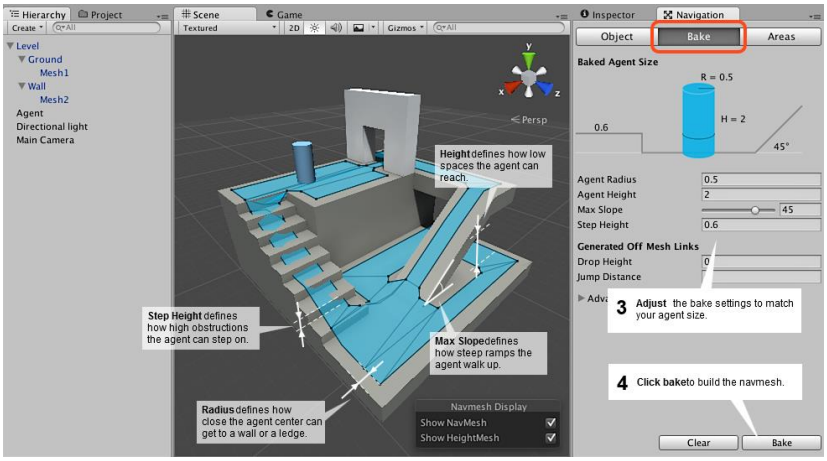
NavMesh Baking – Steps (1,2)



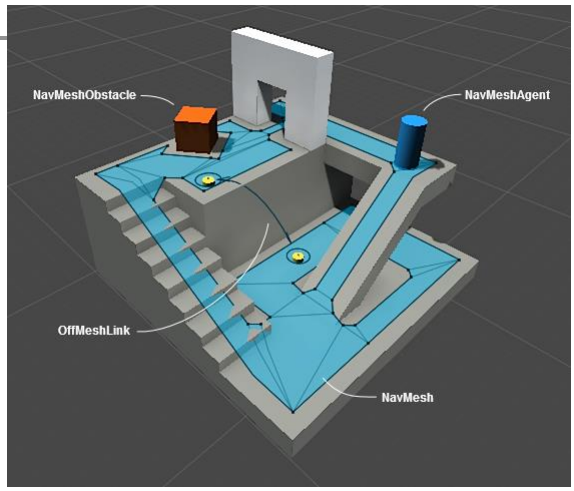
NavMesh Baking Steps

- **3. Adjust the bake settings** to match your agent size.
  - ***Agent Radius*** - how close the agent center can get to a wall or a ledge.
  - ***Agent Height*** - how low the spaces are that the agent can reach.
  - ***Max*** *Slope* defines how steep the ramps are that the agent walk up.
  - ***Step*** *Height* defines how high obstructions are that the agent can step on.
- **4. Click bake** to build the NavMesh.

NavMesh Baking – Steps (3,4)

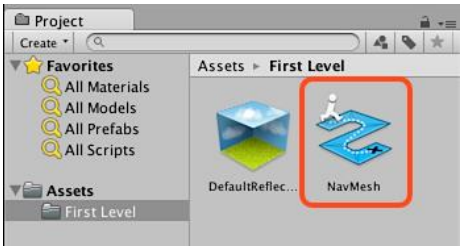


- The resulting **NavMesh** will be shown in the scene as a **blue overlay** on the underlying level geometry.



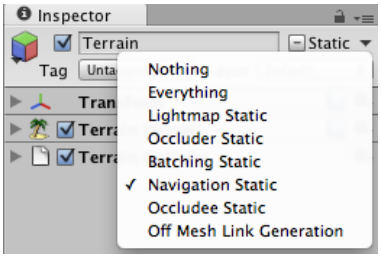
NavMesh Asset

- When baking is complete, you will find a NavMesh asset file inside a folder with the same name as the scene the NavMesh belongs to.



NavMesh Baking– Alternate Workflow

- Select an object and use the ***Static*** menu at the top of the inspector to mark objects as "***Navigation Static***".
  - You don't happen to have the Navigation Window open.



See Advanced NavMesh Bake Settings at https://docs.unity3d.com/Manual/nav-AdvancedSettings.html

Unity Navigation System

- The Unity Navigation system consists of the following pieces:
  - NavMesh
  - **NavMesh Agent**
  - NavMesh Obstacle
  - Off-Mesh Link

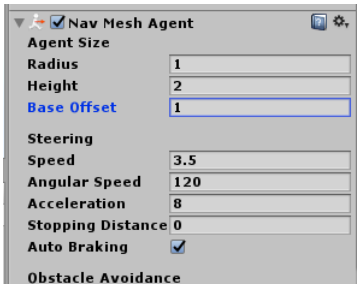
Creating a NavMesh Agent

- Select a character which can navigate the scene.
  - For test, create a **cylinder** via **GameObject > 3D Object > Cylinder**.
- Add a **NavMesh Agent** component to the character via **Add Component > Navigation > NavMesh Agent**.

NavMesh Agent

- Most likely are need to adjust NavMesh Agent's dimensions for your character size and speed.

Radius	Radius of the agent, used to calculate collisions between obstacles and other agents.
Height	The height clearance the agent needs to pass below an obstacle overhead.
Base offset	Offset of the collision cylinder in relation to the transform pivot point (usually center).



For more details, see https://docs.unity3d.com/Manual/class-NavMeshAgent.html

Hands-on Activity

- Create a script (MoveTo.cs) and assign it to the character(cylinder).
- Create a **sphere**, this will be the destination the agent will move to.
- Place the sphere away from the character to a location that is close to the NavMesh surface.
- Assign the Sphere to the **Goal** property of the MoveTo.
- **Press Play**; you should see the agent navigating to the location of the sphere.

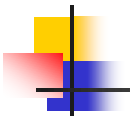
Hands-on Activity

- Create a script (MoveTo.cs) and assign it to the character(cylinder).
- Create a **sphere**, this will be the destination the agent will move to.
- Place the sphere away from the character to a location that is close to the NavMesh surface.
- Assign the Sphere to the **Goal** property of the MoveTo.
- **Press Play**; you should see the agent navigating to the location of the sphere.

MoveTo.cs

```
// MoveTo.cs
using UnityEngine;
using System.Collections;

public class MoveTo : MonoBehaviour {

  public Transform goal; // destination

  void Start () {
    // get a reference to the NavMesh Agent component
    NavMeshAgent agent = GetComponent<NavMeshAgent>();
    // set the agent in motion
    agent.destination = goal.position;
  }
}
```

Use NaveMeshAgent.desitnation to set the destination of the agent in world-space units.



Hands-on Activity

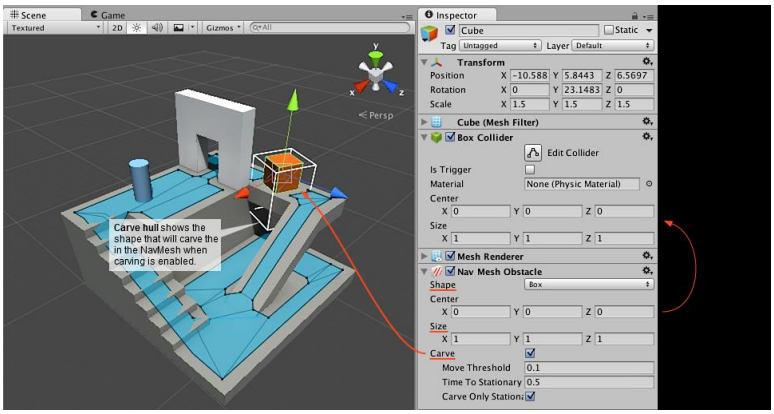
- Create a script (MoveTo.cs) and assign it to the character(cylinder).
- Create a **sphere**, this will be the destination the agent will move to.
- Place the sphere away from the character to a location that is close to the NavMesh surface.
- Assign the Sphere to the **Goal** property of the MoveTo.
- **Press Play**; you should see the agent navigating to the location of the sphere.

Unity Navigation System

- The Unity Navigation system consists of the following pieces:
  - **NavMesh**
  - **NavMesh Agent**
  - **NavMesh Obstacle**
  - **Off-Mesh Link**

NavMesh Obstacle

- Obstacles the agents should avoid while navigating



NavMesh Obstacle – Step 1

- Select a dynamic obstacle in the scene or create a **cube** to depict the crate and place it in the scene.

* If the obstacle does not have a Rigidbody component, add a **Rigidbody Component** to the obstacle.
  - **Add Component** > **Physics > Rigid Body**.

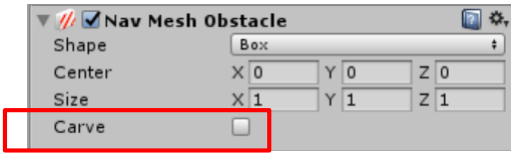
NavMesh Obstacle – Step 2

- Add a "**NavMesh Obstacle"** component to the selected object via **Add Component** > **Navigation > NavMesh Obstacle**.
  - Set the shape of the obstacle to **box**, changing the shape will automatically fit the center and size to the render mesh.
  - **Turn on the Carve** setting from the NavMesh Obstacle inspector so that the agent knows to find a path around the obstacle.

* Now we have a working crate that is physics controlled, and which the AI knows how to avoid while navigating.

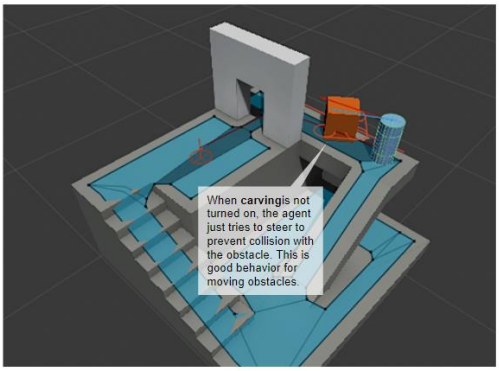
NavMesh Obstacle–"Carve" Off

- When carving is **NOT turned on**, the default behavior of the obstacle is a lot like physics collider.
- The agents will try to avoid collisions with the obstacle, and when close they will collide with the obstacle.



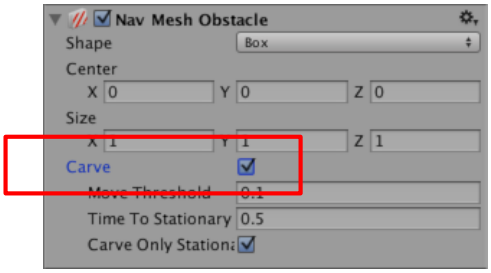
NavMesh Obstacle–"Carve" Off

- This mode is best used in cases where the obstacle is **constantly moving**, for example a vehicle, or even player character.



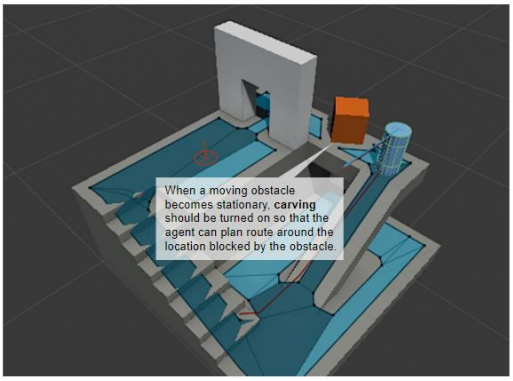
NavMesh Obstacle–"Carve" On

- When "Carve" is turned on, the obstacle becomes **stationary**.



NavMesh Obstacle–"Carve" On

- When "Carve" is turned on, it will carve a hole in the NavMesh so that the agents can change their paths to steer around it.

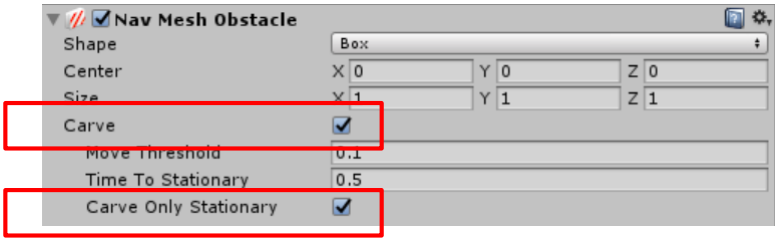


Moving NavMesh Obstacle

- When the Nav Mesh Obstacle moves, the carved hole also moves.
- When to recalculate holes:
  - Only carve when the Nav Mesh Obstacle is stationary
  - Carve when the Nav Mesh Obstacle has moved

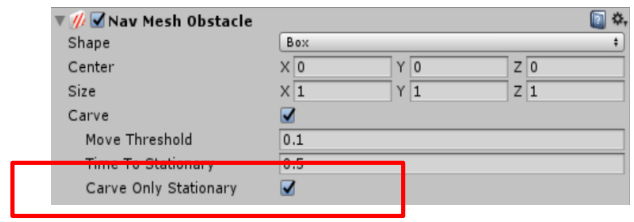
Moving NavMesh Obstacle

- *Carve Only Stationary* - When enabled, the obstacle is carved only when it is stationary.



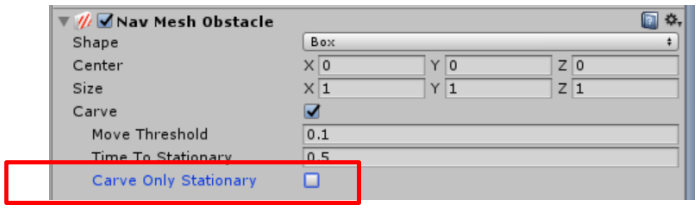
NavMesh Obstacle–"Carve" On

- *Carve Only Stationary is ON.* (this is default)
  - When the Nav Mesh Obstacle has stopped moving and has been stationary for more than the time set by **Carving Time To Stationary**, it is treated as stationary and the carved hole is updated again.
  - It is good match when the game object is controlled by physics (for example crates and barrels).



NavMesh Obstacle–"Carve" On

- *Carve Only Stationary* is *OFF.*
  - The carved hole is updated when the obstacle has moved more than the distance set by **Carving Move Threshold**.
  - This mode is well suited for large slowly moving obstacles.

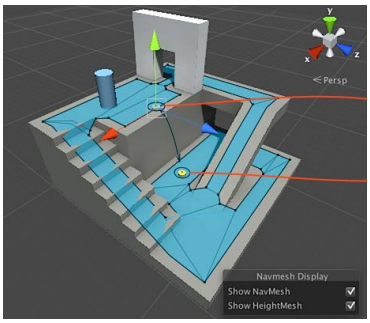


Unity Navigation System

- The Unity Navigation system consists of the following pieces:
  - **NavMesh**
  - **NavMesh Agent**
  - **NavMesh Obstacle**
  - **Off-Mesh Link**

Off-mesh Links

- Off-Mesh Links are used to create paths crossing outside the walkable navigation mesh surface.
  - For example, jumping over a ditch or a fence, or opening a door before walking through it, can be all described as Off-mesh links.
- Let's add an Off-Mesh Link component to describe a jump from the upper platform to the ground.

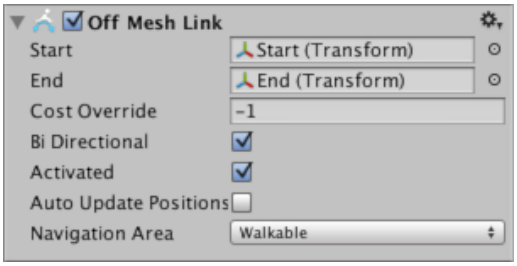


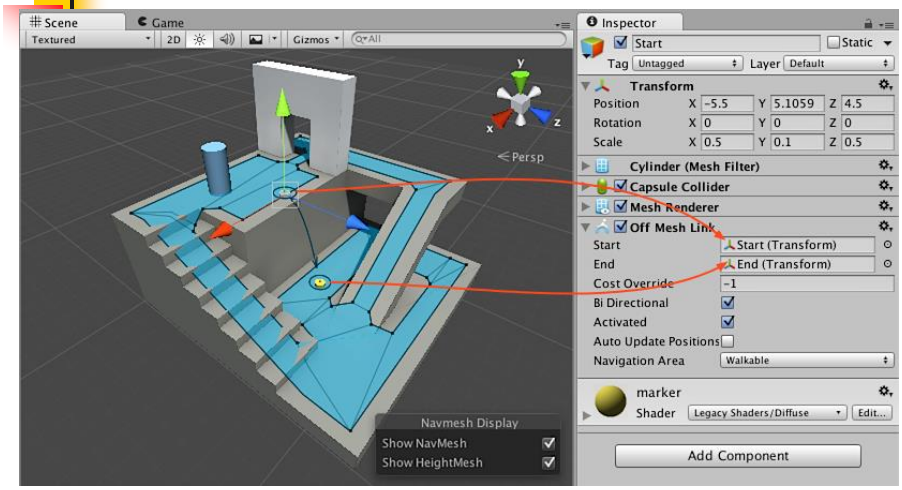
Off-mesh Links – Step 1

- Create **two cylinders**: **Game Object > 3D Object > cylinder**.
  - Move the **first cylinder** at the edge of the top platform, close to the NavMesh surface.
  - Place the **second cylinder** on the ground, close to the NavMesh, at the location where the link should land.

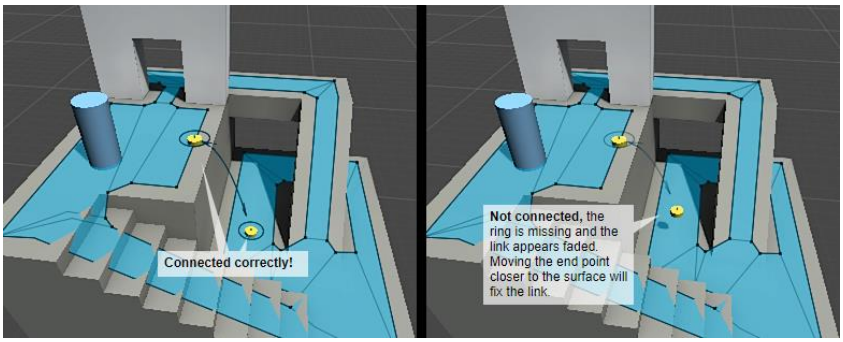
Off-mesh Links – Step 2

- Select the first cylinder and add an Off-Mesh Link component to it. **Add Component** > **Navigation > Off Mesh Link**.
  - Assign the first cylinder in the **Start** field and second cylinder in the **End** field.





If the agent does not traverse an Off-Mesh link, check if both end points are connected correctly. A properly connected end point should show a circle around the access point.



Automatic Off-mesh Links

- Off-Mesh Links can be detected automatically.
- See how to do it at https://docs.unity3d.com/Manual/nav-BuildingOffMeshLinksAutomatically.html

Navigation Control Sample Codes

- A set of techniques and code samples to implement common tasks in navigation.
  - MoveTo.cs
  - MoveToClickPoint.cs
  - Patrol.cs
  - LocomotionAgent.cs

Moving an Agent to a Position Clicked by the Mouse

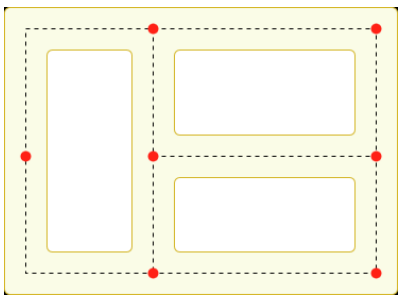
- **MoveToClickPoint.cs** script lets you choose the destination point on the NavMesh by clicking the mouse on the object's surface.
- In Start(), the script stores NavMeshAgent component.
- In Update(), it repeatedly updates the destination based on the clicked location. The position of the click is determined by a *raycast*.

MoveToClickPoint.cs

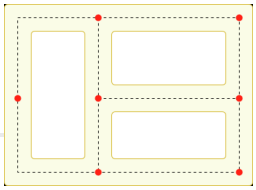
```
// MoveToClickPoint.cs
using UnityEngine;

public class MoveToClickPoint : MonoBehaviour {
  NavMeshAgent agent;

  void Start() {
    agent = GetComponent<NavMeshAgent>();
  }
  void Update() {
    if (Input.GetMouseButtonDown(0)) {
      RaycastHit hit;
      if(Physics.Raycast(
              Camera.main.ScreenPointToRay(Input.mousePosition),
              out hit, 100)) {
        agent.destination = hit.point;
      }
    }
  }
}
```

Making an Agent Patrol

- An Agent patrol between a set of points. Many games feature NPCs that patrol automatically around the playing area.



Making an Agent Patrol



- Set a patrol pattern by keeping a set of **key points** and visiting them in some kind of sequence.
  - For example, in a maze, you might place the key patrol points at junctions and corners to ensure the agent checks every corridor.
  - For an office building, the key points might be the individual offices and other rooms.
  - The ideal sequence of patrol points will depend on the way you want the NPCs to behave.

Patrol.cs

```
// Patrol.cs
using UnityEngine;
using System.Collections;

public class Patrol : MonoBehaviour {

  public Transform[] points;
  private int destPoint = 0;
  private NavMeshAgent agent;

  void Start () {
    agent = GetComponent<NavMeshAgent>();

    // Disabling auto-braking allows for continuous movement between points
    // (ie, the agent doesn't slow down as it approaches a destination point).
    agent.autoBraking = false;

    GotoNextPoint();
  }
```

Patrol.cs (continued)

```
void GotoNextPoint() {
  // Returns if no points have been set up
  if (points.Length == 0)
    return;

  // Set the agent to go to the currently selected destination.
  agent.destination = points[destPoint].position;

 // Choose the next point in the array as the destination, cycling to the start if necessary.
  destPoint = (destPoint + 1) % points.Length;
}

void Update () {
  // Choose the next destination point when the agent gets close to the current one.
  if (agent.remainingDistance < 0.5f)
    GotoNextPoint();
  }
}
```


Patrol.cs Explained

- The patrol `points` are supplied to the script using a public array of `Transforms[]`.
  - This array can be assigned from the inspector using GameObjects to mark the points' positions.
- The `GotoNextPoint` function sets the destination point for the agent (which also starts it moving) and then selects the new destination that will be used on the next call.
  - As it stands, the code cycles through the points in the sequence they occur in the array but you can easily modify this, say by using `Random.Range` to choose an array index at random.

Moving Agent with Animation

- The character moved to the point in the world where the user has clicked on the screen.
  - See `MoveToClickPoint.cs`.
- Let's match the animations with the movement. We need to communicate the state and velocity of the agent to the animation controller.
  - Attach `LocomotionAgent.cs` to the agent.

MoveToClickPoint.cs

```
// MoveToClickPoint.cs
using UnityEngine;

public class MoveToClickPoint : MonoBehaviour {
  NavMeshAgent agent;

  void Start() {
    agent = GetComponent<NavMeshAgent>();
  }
  void Update() {
    if (Input.GetMouseButtonDown(0)) {
      RaycastHit hit;
      if(Physics.Raycast(
            Camera.main.ScreenPointToRay(Input.mousePosition),
            out hit, 100)) {
        agent.destination = hit.point;
      }
    }
  }
}
```

LocomotionAgent.cs

```
// LocomotionAgent.cs
using UnityEngine;

[RequireComponent (typeof (NavMeshAgent))]
[RequireComponent (typeof (Animator))]

public class LocomotionAgent : MonoBehaviour {
    Animator anim;
    NavMeshAgent agent;

    void Start ()
    {
        anim = GetComponent<Animator> ();
        agent = GetComponent<NavMeshAgent> ();
     }
```

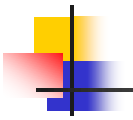
LocomotionAgent.cs

```
     void Update ()
    {
        bool shouldMove = agent.velocity.magnitude > 0.1f &&
             agent.remainingDistance > agent.radius;

        // Update animation parameters
        if(shouldMove) {
            anim.SetBool("Jumping", false);
            anim.SetFloat ("Speed", 0.3f);
        }
        else {
            anim.SetBool("Jumping", false);
            anim.SetFloat ("Speed", 0.01f);
        }

    }
}
```

NavMesh Agent & Others Components

NavMesh Agent & Others Components

- NavMesh Agent and Physics
- NavMesh Agent and Animator
- NavMesh Agent and NavMesh Obstacle
- NavMesh Obstacle and Physics



NavMesh Agent and Physics

- You don't need to add physics colliders to NavMesh Agents for them to avoid each other
  - That is, the navigation system simulates agents and their reaction to obstacles and the static world. Here the static world is the baked NavMesh.
- If you want a NavMesh Agent to push around physics objects or use physics triggers:
  - Add a Collider component (if not present)
  - Add Rigidbody component
    - Turn on kinematic (Is Kinematic) - this is important!
    - Kinematic means that the rigid body is controlled by something else (Navigation system in this case) than the physics simulation

NavMesh Agent and Physics (2)

- If both NavMesh Agent and Rigidbody (non-kinematic) are active at the same time, you have race condition
  - Both components may try to move the agent at the same which leads to undefined behavior

NavMesh Agent and Animator

- NavMesh Agent and Animator with Root Motion can cause race condition
  - Both components try to move the transform each frame
  - Two possible solutions
- Information should always flow in one direction
  - Either agent moves the character and animations follows or animation moves the character based on simulated result
  - Otherwise you'll end up having a hard to debug feedback loop

* NavMesh Agent and Animator without Root Motion – leave the transform to the NavMesh Agent and choose to play a proper animation when the agent moves.

NavMesh Agent and Animator (2)

- **Animation follows agent**
  - Use the NavMeshAgent.velocity as input to the Animator to roughly match the agent's movement to the animations
  - Robust and simple to implement, will result in foot sliding where animations cannot match the velocity
- **Agent follows animation**
  - Disable NavMeshAgent.updatePosition and NavMeshAgent.updateRotation to detach the simulation from the game objects locations
  - Use the difference between the simulated agent's position (NavMeshAgent.nextPosition) and animation root (Animator.rootPosition) to calculate controls for the animations
  - See Coupling Animation with Navigation. https://docs.unity3d.com/Manual/nav-CouplingAnimationAndNavigation.html

NavMesh Agent and NavMesh Obstacle

- Do not mix well!
  - Enabling both will make the agent trying to avoid itself
  - If carving is enabled in addition, the agent tries to constantly remap to the edge of the carved hole, even more erroneous behavior ensues
- Make sure only one of them are active at any given time
  - Deceased state, you may turn off the agent and turn on the obstacle to force other agents to avoid it
  - Alternatively you can use priorities to make certain agents to be avoided more

NavMesh Obstacle and Physics

- If you want physics controlled object to affect NavMesh Agent's behavior
  - Add NavMesh Obstacle component to the object which agent should be aware of, this allows the avoidance system to reason about the obstacle
- If a game object has a Rigidbody and a NavMesh Obstacle attached, the obstacle's velocity is obtained from the Rigidbody automatically
  - This allows NavMesh Agents to predict and avoid the moving obstacle.