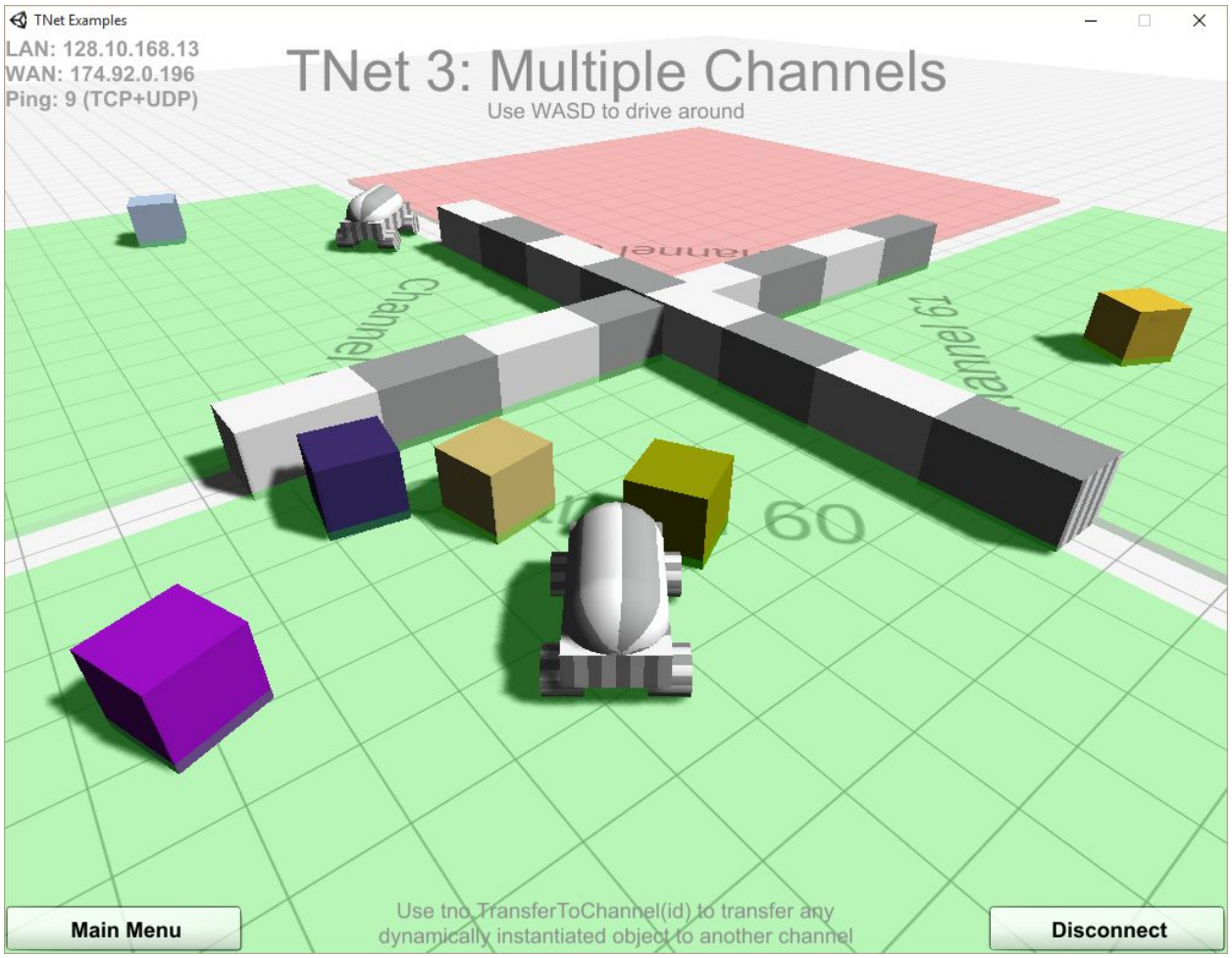
TNet 3: Multiple Channels In-Depth



The Multiple Channels example scene demonstrates how to divide a large scene up into separate channels. This allows the amount of network traffic to be kept to a minimum. If a character is in another town in an open world game, you don't really care what their position is until they venture over to where you are.

Note: This example builds on the Movement example. If you would like to see how the camera tracks the car, the vehicle is spawned, or how it updates its position, please take a look at that document.

The scripts attached to the Car 2 prefab used here are a little different from the car in the movement example. It includes the `ExampleCar` script but it also has a new script called `ExamplePlayerAvatar` which will be detailed shortly.

All other scripts that are included in this example are located on the Channel GameObjects. First let's cover the simple `ExampleCreate` script. It spawns a "Heavy Cube" in the specified channel when the user clicks the floor.

```
public int channelID = 0;
```

`channelID` holds the number of the channel where the cube will be spawned. In the inspector it is set to a number between 60 and 63. `OnClick()` the script picks a random position 3m above the channels floor, and instantiates the cube in that channel.

```
TNManager.Instantiate(channelID, "ColoredObject", prefabName, true, pos, rot,
color, autoDestroyDelay);
```

The cubes are simple this way, they spawn in a channel and even if they tumble into another one, they will always exist in that particular channel. The car however, will move and it needs to know about the various areas of the map. To do so each Channel GameObject has an `ExampleRegion` script.

```
static public List<ExampleRegion> list = new List<ExampleRegion>();
```

When it's first created, a static list of all regions is created. This allows for quick searches to be performed later.

```
void OnEnable ()
{
	list.Add(this);
	TNManager.onJoinChannel += OnJoinChannel;
	TNManager.onLeaveChannel += OnLeaveChannel;
}

void OnDisable ()
{
	list.Remove(this);
	TNManager.onJoinChannel -= OnJoinChannel;
	TNManager.onLeaveChannel -= OnLeaveChannel;
}
```

When enabling and disabling this region is added or removed from the region list, and at the same time event callbacks are registered to the local client. If the client joins or leaves a channel, the script toggles the colour of the material between red and green.

```
Color c = TNManager.IsInChannel(channelID) ? Color.green : Color.red;
```

Now that the regions are defined in a searchable list, the car can start making some decisions for the local client and which channels it belongs to. Take a look at the `ExamplePlayerAvatar` script on the "Car 2" prefab.

```
public float joinDistance = 14f;
public float leaveDistance = 16f;
```

It starts off by defining two circles. The Join circle is 14m in radius and the leave circle has a 16m radius. By making the "leave" distance greater than the "join" distance, any loading and unloading of meshes will happen on different frames and reduce any reduction in the player's frame rate.

```
IEnumerator Start ()
{
	if (tno.isMine)
	{
		// Wait until we've joined the channel before starting the periodic
	checks
		while (TNManager.isJoiningChannel) yield return null;
		InvokeRepeating("PeriodicCheck", 0.001f, 0.25f);
	}
	else Destroy(this);
}
```

On Start, if this car belongs to this client, check to see if we're in a channel. If we are, start calling the `PeriodicCheck()` function every quarter of a second. If this car doesn't belong to us, but is instead a networked car, remove this script off of it because it doesn't need to be here.

```
// First find the closest region -- this is the region the player avatar should belong to
for (int i = 0; i < ExampleRegion.list.size; ++i)
{
	ExampleRegion region = ExampleRegion.list[i];
	float distance = Vector3.Distance(region.transform.position, myPos);

	if (distance < closestDistance)
	{
		closestDistance = distance;
		closestRegion = region;
	}
}
```

`PeriodicCheck` first loops through the entire list of regions in the world and looks to see which region is closest to the car. The closest one is recorded and it moves on.

```
// Now ensure we've joined all the nearby regions in addition to the closest
region
for (int i = 0; i < ExampleRegion.list.size; ++i)
```

```
{
	ExampleRegion region = ExampleRegion.list[i];
	float distance = Vector3.Distance(region.transform.position, myPos);

	if (distance < joinDistance || region == closestRegion)
	{
		// We're close -- join the region's channel
		if (!TNManager.IsInChannel(region.channelID))
			TNManager.JoinChannel(region.channelID, true);
	}
	else if (distance > leaveDistance && tno.channelID != region.channelID)
	{
		// We're far away -- leave the region's channel
		if (TNManager.IsInChannel(region.channelID))
			TNManager.LeaveChannel(region.channelID);
	}
}
```

It loops through all of the region's again, but this time when it does the distance calculation it does a couple checks on it. First it looks to see if it is within the join (14m) distance or if it's the closest region. If it is, it makes sure that the local client is in the channel belonging to that region.

If however the distance is greater than the leave distance and the car doesn't belong to that channel, then tell the client to leave the channel. The check on the car is there to make sure that the players car never accidentally deletes itself.

```
// Transfer the car to the closest region's channel
if (closestRegion != null && tno.channelID != closestRegion.channelID)
	tno.TransferToChannel(closestRegion.channelID);
```

And finally, if the closest region is different, it will transfer channel ownership of the car over to that region's channel.