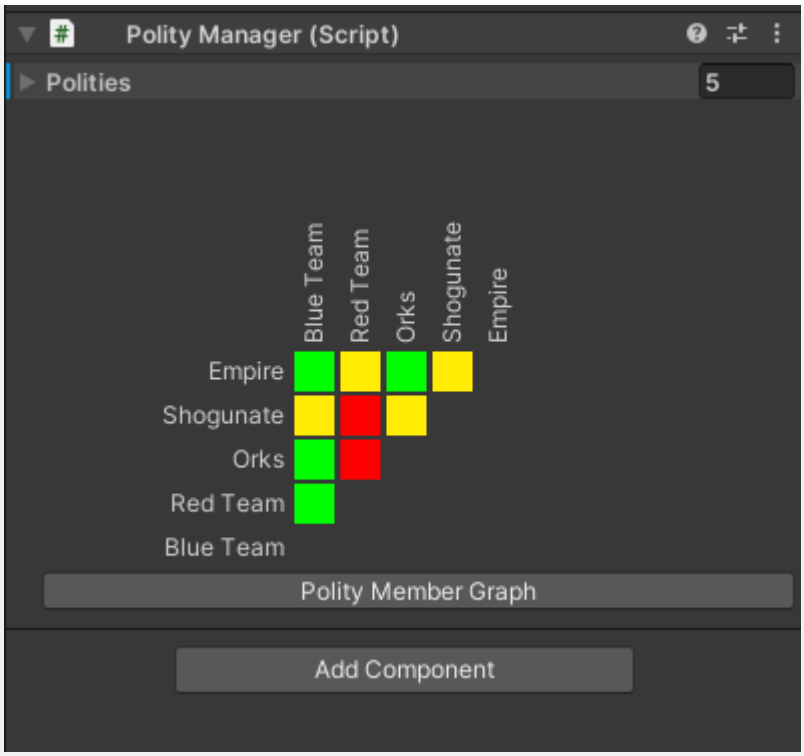
Polity Manager - Manage Factions, Teams & Families

Description

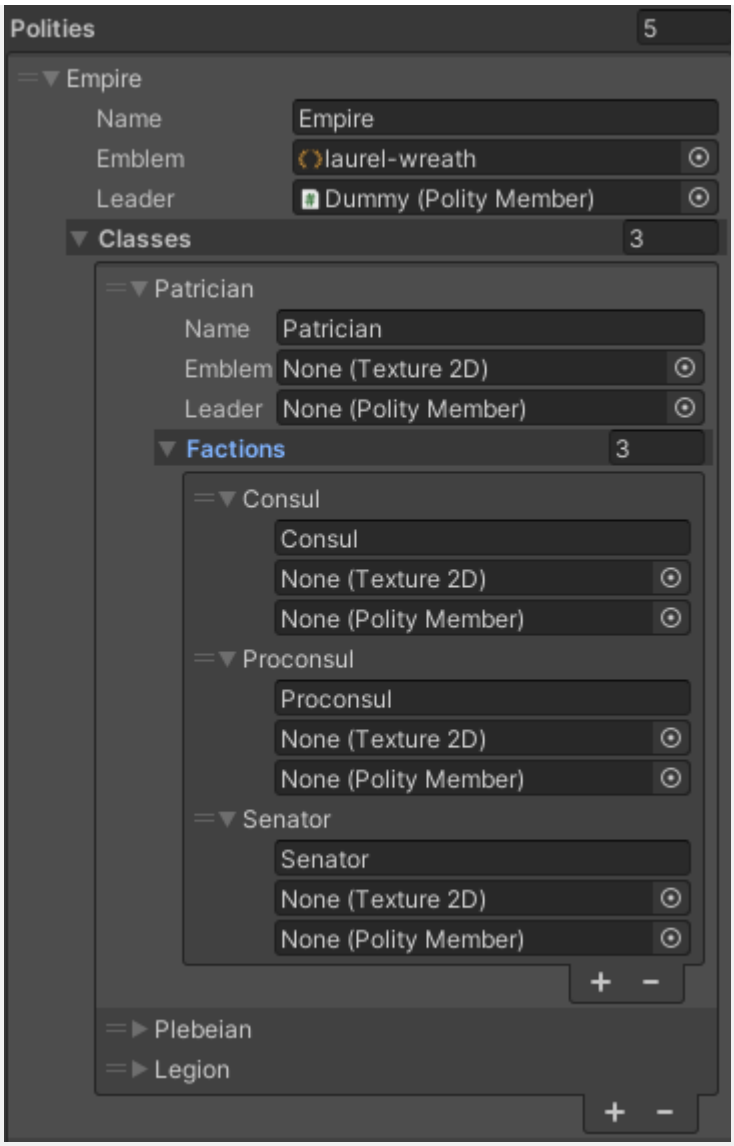
Polity Manager is an editor based tool designed to manage relations between polities in a centralized matrix, along with individual family relations through a simple node graph.

The PolityManager singleton contains a *Polity Relation Matrix*, a grid table that displays the relation of one polity to another based on their matrix position, similar to the Unity physics collision matrix.



The Red Team is allied to the Blue Team, but are enemies to the Orks and the Shogunate.

A Polity[1] also contains a serialized array of Class[2] objects, and each Class object has a List of Faction[3] objects. These serve to departmentalize the various branches or groups of your polity into smaller, more manageable units.



Each political unit, from the Polity to Faction, can be given a name, Texture2D emblem and a PolityMember leader.

To connect these polities to a prefab GameObject, the `PolityMember.cs` component is attached to that GameObject which will now assign it to a created polity, along with their class and faction (note that the class and faction will create an empty selection first, in case the PolityMember does not want to be affiliated with a class or faction).

The Shogunate polity has a Daimyo class, and that class contains a faction called the Nissan Clan.

Polity Manager is suited for games that needs to manage various groups of NPCs, especially when these relationships are a bit more complex, such as when one NPC needs to react to an enemy of one or more allied NPCs. However, it can also be applicable to simple teams or solo duels.

Quickstart

Video Tutorial

Demo Tutorial

1. Play the Example Demo.unity Scene
2. Click on the PolityManager GameObject in the hierarchy
3. Click on the grid cells in the *Polity Relation Matrix* to change RelationType[4] between polities.
    - NPCs will react by targeting enemies or enemies of allies.
4. Hover your mouse over an NPC to view their selected polity and family.

This should demonstrate a very basic implementation of how the PolityManager can control NavMeshAgents with a PolityMember that can react to relationship changes based on their current polity.

You can open the `NPC_Driver.cs` class inside of Example/Scripts to get a better idea of how the class subscribes to events and how it calls public PolityManager methods.

Public APIs

All classes in this package is under the `KhiemLuong` namespace.

PolityManager.cs

All public methods can be called from this PolityManager Singleton, referenced as `PM`, for example PM.ModifyPolityRelation();

ModifyPolityRelation()

Sets a new relationship status between two polities based on their names, adjusting their relation to either Neutral, Allies, or Enemies. If the polities matched, the `OnRelationChange` event will be invoked to notify all subscribers of the relation change.

Parameter	Type	Description
`politymember`	`PolityMember`	The member of the polity initiating the relationship change.
`theirPolityName`	`string`	The name of the polity that is targeted for the relationship change, retrieved from `politymember` in `PolityMember`.
`factionRelation`	`PolityRelation`	The new relation to set; can be `Neutral`, `Allies`, or `Enemies`.

GetPolityRelation()

Compares the PolityRelation of the politymember to the second politymember and returns the enum that indicates their PolityRelation if found. The overload method replaces the PolityMember parameters as strings representing the politymember.

Parameter	Type	Description
`politymember`	`PolityMember`	The member of the polity initiating the relationship comparison.

Parameter	Type	Description
`theirPolityMember`	`PolityMember`	The name of the polity that is targeted for comparison, retrieved from `polityName` in `PolityMember`.

Returns The `PolityRelation` enum value, which can be `Neutral`, `Allies` or `Enemies`.

GetPolityEmblem()

Gets the current Texture2D emblem of the political unit which you want, depending on whether you provide more than just the polityName. Meaning if you also provide a className and factionName, it will return that faction's emblem instead.

Parameter	Type	Description
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, and optionally a class and faction.

Returns The `Texture2D` of that political unit's emblem

GetPolityLeader()

Gets the current leader of the political unit which you want, depending on whether you provide more than just the polityName. Meaning if you also provide a className, it will return that class leader instead.

Parameter	Type	Description
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, and optionally a class and faction.

Returns The `PolityMember` of that political unit's leader

AddFactionToPolity()

Determines the polity and class based on the PolityStruct parameter, and searches through the factions List inside that polity's class, adding one if a faction was not found. A Texture2D and PolityMember can be given as overload parameters to set it as the new faction's emblem and leader.

Parameter	Type	Description
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, className and factionName.
`emblem`	`Texture2D`	The image which the new Faction's emblem will be.
`leader`	`PolityMember`	The leader of this new Faction.

RemoveFactionFromPolity()

Removes a faction of a polity, if the PolityStruct polityName, className and factionName all match.

Parameter	Type	Description
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, className and factionName.

Returns The `PolityRelation[,]` matrix which was deserialized from the string.

SerializePolityRelationMatrix()

Serializes the `PolityRelation[,]` matrix and sets the serialized `polityRelationMatrixString` to its return value.

Parameter	Type	Description
`polityRelationMatrix`	`PolityRelation[,]`	The 2D PolityRelation matrix. This can be omitted which will use the Singleton's `polityRelationMatrix`.

Returns The `string` of that serialized matrix.

DeserializePolityRelationMatrix()

Deserializes a string representing the `PolityRelation[,]` matrix.

Parameter	Type	Description
`json`	`string`	This can be omitted which will then use the Singleton's `polityRelationMatrixString`.

Returns The `PolityRelation[,]` matrix which was deserialized from the string.

PolityMember.cs

A non-serialized MonoBehaviour class which communicates with the PolityManager Singleton to retrieve its polities, classes and factions which can be set for the specific GameObject that it is attached to.

ChangeMemberPolity()

Changes the current PolityMember's polity, class and faction based on what parameters were provided.

Parameter	Type	Description
`_struct`	`ref PolityStruct`	The PolityStruct which must include a polityName, and optionally a class and faction. Passed in as a `ref` so the change is applied to the PolityMember.

SetAsPolityLeader()

Sets a new leader for the polity, or its class and faction to the PolityMember which called the method.

Parameter	Type	Description
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, and optionally a class and faction.

static SetPolityLeader()

Sets a new leader for the polity, or its class and faction to the `PolityMember` newLeader parameter.

Parameter	Type	Description
`newLeader`	`PolityMember`	The PolityMember which will be the new leader of this polity. Set to `null` if you want this polity to have no leaders.
`_struct`	`PolityStruct`	The PolityStruct which must include a polityName, className and factionName.

static GetMembersInPolity()

Gets all `PolityMember[]` classes in the scene, then filters out the members which belongs to the polity, class or faction according to the provided `PolityStruct`.

Parameter	Type	Description
`_struct`	`PolityStruct`	Gets the `PolityMember[]` based on the provided struct values (polityName, className, factionName)
`getInactive`	`boolean`	Includes inactive PolityMember[] if true. defaults to false in overload if not provided.

Returns The array of filtered PolityMember[] belonging to the PolityStruct.

GetMemberPolity()

Returns The `PolityMember`'s `PolityStruct`.

GetMemberFamily()

Returns The `PolityMember`'s `FamilyStruct`.

Events

OnRelationChange

Invoked whenever `ModifyPolityRelation()` is called or when the *Polity Relation Matrix* cell is clicked on.

OnLeaderChange

Invoked whenever a `PolityMember` is set to a new Polity as their leader with `SetAsPolityLeader()` or `SetPolityLeader()`.

OnFactionChange

Invoked whenever a `Faction` is created or removed with `AddFactionToPolity()` and `RemoveFactionFromPolity()`.

Structs

PolityStruct

```
public struct PolityStruct
{
   public string polityName;
   public bool isPolityLeader;
   public string className;
   public bool isClassLeader;
   public string factionName;
   public bool isFactionLeader;
}
```

FamilyStruct

```
public struct FamilyStruct
{
   public PolityMember[] parents;
   public PolityMember[] partners;
   public PolityMember[] children;
}
```

PolityManager.cs ContextMenus

Reset Polity Relation Matrix

Reset every relation to `Neutral`.

Find Duplicate Polity Names

Checks the polity names in `Polity[]` array and logs a warning for any duplicate names.

Load Polity Relation Matrix

If in some case the serialized polity relation matrix did not load, this can manually deserialize & load it.

PolityMember.cs ContextMenus

Check Family

Check the relation between each family member to ensure it is reciprocal, otherwise remove it.

Delete Family

Removes all family members and its reciprocal from the `parents`, `partners` & `children` Lists.

Cleanup Family

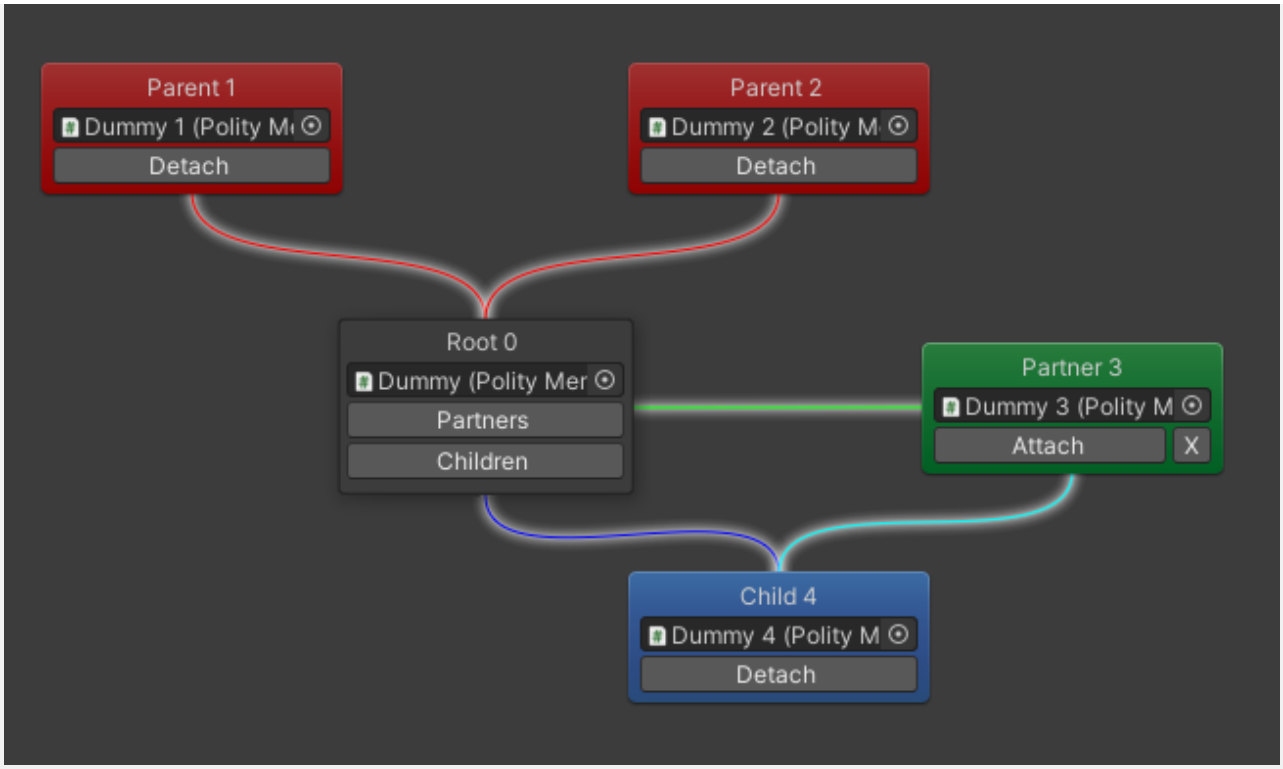
Removes any null or missing PolityMember family from the `parents`, `partners` & `children` Lists.

Extra Settings

Families

You can set a PolityMember to a polity and create a political relation to another polity if this is all you need. However, you can expand this relationship to each individual PolityMember for them to be connected not just by political affiliation, but by family.

The *Member Family Graph* is a node based graph which can be accessed by clicking on the `Member Family Graph` Button under the *Polity Relation Matrix* inside of `PolityManager`. The family structure that is created in this graph is **relative**, meaning that it will only include the Root Node's parents, partners, and children which is directly related to the root node.



The root node (Root 0) indicates 2 parents indicated by a red line with red nodes, a partner indicated by a green line with green nodes, and their child, with the blue line indicating lineage from the root node, and the cyan line from the root node's partner with that child.

The result is a family lineage that is separate from the political affiliations of that PolityMember. Meaning that it could be possible for a parent or child of a PolityMember to be enemies, despite being related.

Root Node

When the *Polity Member Graph* is opened, it will render one node, the Root Node, which requires you to place in a prefab with a PolityMember component attached to it. Once you have placed in a PolityMember, it will display 3 buttons: Parents, Partners and Children.

You can now add more nodes to this graph by clicking `Add Node` to the top left of the window, which will create more nodes that also needs a PolityMember prefab reference. However, these nodes are different in that they instead only have a button called Attach.

You can assign relationships to these other nodes from the Root Node by clicking the respective button, such as **Parents** if you want to create a parent relationship to that node, then the Attach button to the node you want to make that **Parent** relationship to.

While it is not necessary to have 2 parents, there can only have 2 parents per node. However, the root node can have as any partners as possible, and as many children with any of those partners.

Limitations

Since the rendered nodes only display immediate family members to the root node, not the entire family tree from the first descendant, there is the possibility of accidentally reassigning the same PolityMember to a more distant but related family member.

Therefore, this *Polity Member Graph* should **not** be used as a solution for managing and building complex and large family relationships. You should rely on third party software to design your family trees, then use the graph to assign the relationships according to that design.

Another factor to consider is having multiple instances of a prefab with a defined family structure, which could cause confliciting references. Generally, you should not spawn more than one `PolityMember` NPC that belongs to a family, and these NPCs should be treated as unique essential NPCs. Conversely, `PolityMembers` who do not belong to any family, or generic NPCs, can be spawned multiple times without issue.

Leaders

Every `Polity`, `Class` and `Faction` object is inherited from the abstract `PolityBase` object, which contains a `string` name, `Texture2D` emblem, and `PolityMember` leader. You can assign a prefab with a PolityMember attached to this leader variable like how you would with a *Polity Member Graph* node. This will then mark the respective polity, class or faction of that PolityMember to be a leader of that political unit, denoted by "|Leader|".

This PolityMember belongs to the `Orks` polity, which has a `Gretchin` class, of which it is a leader of. Note that since the `Gretchin` class does not have any factions, the `Faction` dropdown does not appear. Also note that it has a child assigned from the graph, who belongs to a different polity.

Credits

PBR Sand054 texture ambientcg.com - CC0 License

Polity Manager was developed by Khiem Luong (github.com/khiemgluong)

Glossary

1: Polity - Represents the largest & most important political unit such as a government body, corporation or main team.
2: Class - Represents a social class, government branch, organization, or any large collective corp.
3: Faction - Represents a small and temporary political unit, which can be added and removed at runtime.
4: RelationType - an enum value representing the current relationship between 2 polities; declared as Neutral, Allies and Enemies.