**ARLearn so sinnvoll? Erstmal am besten nur sandboxig rein, was rein soll (3-4 Sachen, dann gucken, was die Zeit sagt)**

* Entweder den User switchen lassen, was aktiv ist, oder einfach nebeneinander klatschen

Ansatz (1. Version?):

Idee: ~~I\_Targetable vererbt an I\_MovementTargetable und I\_GazeTargetable~~ – nicht so super, da nur Focused als Gemeinsames Sinn macht

Alternative Idee: Für das Verhalten „Ruft Event auf, destroyed oder disabled sich und implementiert UI-Zeug“ erben DirectedFocusTarget und DirectedMovementTarget von einer DirectedTarget Klasse

Pattern: Directed Focus  
Klassen: GazeManager, DirectedFocusTarget implementiert I\_GazeTargetable und I\_UiPointer  
Funktion:   
Gaze Manager: Sendet jeden Frame Raycasts aus, sendet Focused Event an getroffenes I\_GazeTargetable Objekt  
DirectedFocusTarget: Focused(): Ruft UnityEvent (?) auf, destroyed oder deaktiviert sich dann. Wenn nicht im Bild: Pfeil im UI (Bei dem einen HoloToolkit Beispiel gucken).

Pattern: MovementManager, DirectedMovementTarget implementiert I\_UiPointer und I\_MovementTargetable  
Funktion:   
MovementManager: Macht raycasts nach unten, sendet Moved Event an I\_MovementTargetable   
DirectedMovementTarget: Moved(): Ruft UnityEvent (?) auf, destroyed oder deaktiviert sich dann, auch hier UiPointer-Zeug

Grundelement (evtl.) Struktur, die man speichern und laden kann

Ansatz von ARLearn:

* “Media such as video, audio, open questions and multiple choice question can be bound to a location, time or game action.”
* “ARLearn features an open architecture as it can easily be extended with new features and enables third party applications to integrate with ARLearn by implementing XMPP based listeners. Extensible Messaging and Presence Protocol (XMPP) is a protocol for real-time messaging. . . . The ARLearn architecture features a REST (Representational state transfer) API facilitating the creation of mash-ups enabling developers to interact with the ARLearn services.”
* 
* “Within a game, an author defines items, dependencies between items, game score rules and progress rules. Once a game has been created, an arbitrary amount of runs can be created and played. A run defines users grouped in teams. While users play a run, they generate actions (e.g. “reading a message”, “answering a question”) and responses. This output is also managed within the realm of a run.”
* “Two communication protocols for clients permit retrieving information from ARLearn. The REST based API features basic CRUD (Create, Read, Update and Delete) operations. For instance, a game can be created by posting an XML or a JSON representation to the service endpoint.”
* 
* “At the core of ARLearn are media items that hold information or add a function to the game. In general, media items can be positioned on a map by providing latitude and longitude attributes. Not providing location attributes turns the item into message that users can receive at some point in the game. Through defining a multiple-choice item, users can answer questions with predefined answers. Audio feeds are integrated via audio objects with some explanatory text. Apart from information items, dynamic items such as a transport task let users perform actions. A transport task defines a pickup item and a dropzone. A pickup item can be taken by users and can be brought to a location where it is dropped. Dropping it at the correct dropzone can lead (through dependencies) to new available items, increased scores or increased game progress.”
* “Generic items have a simple life-cycle with three states: At the launch of a run, an item can be either visible or invisible. When an item is invisible, it can become visible. Later, when the item is no longer needed, it can become invisible again. Items can define a dependsOn and a disappearOn attribute to define the condition for the item to become respectively visible and invisible. A simple dependency mechanism is put in place to support these conditions.
  + An action-based dependency becomes true once a certain game action has been triggered. For instance, the action “startRun” is triggered when the user starts a run for the first time. Alternatively, one can make an item depend on opening another item (“read” action). Scope can take three values: “user”, “team” and “all”. Making an item depending on a “read” action with scope all, will render the item visible as soon as one player in the run performs the “read” operation. Scope “team” will render the item visible if and only if a player in the same team opens the item and “user” requires the user to open the item first.
  + A time-based dependency binds a time offset to another dependency. The dependency is fulfilled when two conditions are completed: the referenced dependency must be completed and the time must have elapsed with the amount of milliseconds specified by the offset attribute. For instance, if a message must be show to the user 10 minutes after the run has been started, a time-based dependency is created that refers to an action (startRun) based dependency with an offset of 600.000 milliseconds.
  + ● Boolean dependencies provide a means to create “AND” and “OR” statements with other dependencies.”
* 

Unterschiede hier:

* Teameinbindung sinnvoll? (Scope dependency und scope der Bachelorarbeit)
* Lokale Koordinaten (Anchors) (bei gleichen Orten), bzw. Queries bei Umgebungsunabhängigkeit statt Weltkoordinaten
* Mehr Game-Möglichkeiten als Multiple Choice (API , mit dem Unity wie auch immer kommunizieren kann)
* Komplexere Interaktionen nötig -> Kern der BA -> Evtl. nur das modellieren (Rahmensprengung)  
  z.B.: GGV, immer noch Position

MACHT BASIS AUF ARLEARN ÜBERHAUPT SINN?

ICH WEISS GAR NICHT WIE DAS TATSÄCHLICH AUSSIEHT

Sagen wir:

* XML-Datei / Scripting Language
* BEDINGUNGEN:
  + POSITION
  + TIME
  + GAZE
  + GESTURE
  + CUSTOM VARIABLEN
* MODIFIER:
  + AND
  + OR
  + NOT
* FUNKTIONEN mit Argumenten
  + E.g. Spawn
  + Change (custom) var
  + Custom funktionen
  + Quit
  + Restart