Chords are here used to denote 1 or more notes being played simultaneously.

This document is meant to get an overview of the separation of activation patterns and the actions performed by objects.

**Activators:**

*Activators* are the parts of objects that the player is able to give input to, and based on the input they get, they will send either an *Activation*, *Deactivation*, and in some cases a *Reset* signal to an *InteractableAction* that it holds a reference to.

Need to extend the Activators so that they can have several InteractableActions.

From what I gather there should be something like these 3 different kinds of *Activators* available to build puzzles with:

SwitchActivator (WIP)

* When a specific chord has been played for a duration it will activate.
* An option to have it deactivate again if the chord is played for a duration
* An option to deactivate automatically after a certain time has passed
* As standard it will use both a pressure sensitivity range and a chord to activate (the range will just be set include the whole spectrum)
* Needs to have a specific duration that it can include in the SongData that it sends (maybe)

ContinuosActivator (WIP)

* While a specific chord is being played it is activated, and when the chord stops being played it deactivates (there is an activation- and deactivation-delay that are adjustable)
* Will give the pressure-sensitivity information to its Action
* Have option to activate when a certain volume is reached (pressure sensitivity) and maintained

SequenceActivator (Not started working yet)

* Has a list of (chord, playTime, deactivationDelay) where if *chord* has been played for *playTime*, it will send an Activation signal, and increment the index for which element in the list it checks
* If nothing has been played for *deactivationDelay* of the current index, the index will decrement and it will send a Deactivation signal
* Whenever the index changes, the miniWheel showing what should be played will also change
* Needs to have a specific duration that it can include in the SongData that it sends (maybe)

**InteractableActions:**

*InteractableActions* can receive *Activation*, *Deactivation,* and *Reset* signals through the *Activate(),* *Deactivate(),* and *Reset()* methods. When they receive these signals it is up to themselves to do something based on these signals.

Available methods:

* Activate()
* Deactivate()
* Reset()
* InputData(SongData data)

If InteractableActions need to be able handle partial activation and deactivation they should handle it internally (e.g. a bridge that only unfolds partially and a sequence is played to unfold each section)

PhysicsAction

* When activated applies a force either UP, LEFT, or RIGHT (can also implement DOWN), multiplied by the pressure of the triggers
* When deactivated, stops applying the force
* When reset returns to its original position

MoveToPositionAction

* Has a list of positions that it can move between
* When it gets activated it starts moving between the positions
* Can toggle whether or not should be able to be deactivated, if deactivated it stops moving
* When reset, it is placed at its original location
* Option to make it move back and forth between its points (and an option to wait a set time at every position before moving again)
* If it has more than 3 positions there is an option to loop, so that once it reaches the last position it will move to the first position, and repeat

RotateAction

* When Activated, rotates for a certain duration
* When Deactivate, rotates the same amount the other direction
* Has option choose clockwise or counterclockwise rotation
* Has a speed with which it rotates
* Option to have pressure affect the rotation-speed

MoveAction

* When Activated moves for a specific duration at a set velocity specified by a vector and a speed variable (it should get the duration from the SongData)
* When Deactivated it stops moving
* When Reset, it moves to its original position
* Option to have pressure affect the move-speed
* Options to either use only the x-, y-, or both components of the vector for movement

LightAction (Not started working yet)

* When Activated will increase light intensity based on the pressure
* When Deactivated, will will fade back to original light intensity over time
* When Reset, will go instantly to original sensitivity

AudioAction (Not started working yet)

* When Activated will playback an audio-clip with a speed determined by the pressure of the trigger
* When Deactivated, will fade out the playing of the audio-clip
* When Reset, will instantly stop the audio-clip and make sure it will start at the beginning next time

DestroyAction (Not started working yet)

* When Activated, will call another script that will handle how this specific objects should be destroyed (like if animation, sound, or particle-effects are wanted)
* When Deactivated, does nothing
* When Reset(), resets the object that was destroyed to it original state

DisableActivatorsAction

(Needed if we want the same notes to map to different actions depending on the situation, like play note ‘A’ to move box upwards, but if volume is above a certain threshold, keep it still in the air instead)

* When Activated, will disable all the activators in a list
* When Deactivated, will enable all the activators in a list
* When Reset(), will call Deactivate()