# Concepts

The application has notion of the following concepts:

The application has notion of a list of architectural **Projects** that can be reviewed. A project can be reviewed using several **NavigationModes**.

A **Project** has notion of a list of **ProjectPhases**. Each phase represents a state of the project at a given point in time.

A **ProjectPhase** has notion of the following:

* a list of **ProjectLayers**
* a list of **POI (Points-Of-Interest)**
* a world bounding box
* a model
* predefined construction lighting

# ApplicationState

The ArchiVR has 2 application states:

* ApplicationStateHome
* ApplicationStatePlay

## Application state design

### Class hierarchy

The class **ApplicationStateBase** is the abstract base class for all concrete application state classes. There are 2 concrete application state classes:

* ApplicationStateHome
* ApplicationStatePlay

The application state class is responsible for managing:

* Available and active **XR Device**
* **UI mode**
* **Graphics Settings**
* **Control Settings**

### XR Device

The following concepts exist:

* Supported XR device
* Available XR device
* Active XR device

#### Supported XR device

A supported XR device is an XR device that is supported by the application. The list of supported XR devices is:

* None
* Split
* Stereo
* Cardboard
* Oculus

In the future, XR devices **Microsoft XR** and **HTC Vive Focus** may be added tot he list of supported XR devices.

The list of available XR devices depends on the build settings of the application. The list is determined dynamically however at application startup, but is constant during the entire execution of the application.

#### Available XR device

An available XR device is a supported XR device that is also supported by the system on which the application is currently running. Not all supported XR devices are effectively supported by all systems:

* **Oculus** XR device is only supported by systems compatible with either GearVR or Oculus headset.
  + GearVR compatible phones with Oculus prerequisite software installed.
  + PC’s in combination with an Oculus headset and Oculus prerequisite software installed.
* **Cardboard** XR device is only supported by systems compatible with Google Cardboard.
  + Mobile systems with gyroscope hardware and Google Cardboard prerequisite software installed.

The list of available XR devices depends on the specifics of the system on which the application is running. The list is determined at application startup:

* If **Oculus** is available on the current system, it is automatically set as the initial active XR device by the Unity runtime. In this case this also becomes the only available XR device: no other XR devices from the supported XR device list are added as tot he available RP device list.
* Otherwise, the following XR devices are added tot he available XR device list:
  + None
  + Split
  + Stereo
  + Cardboard (if supported?)

The list of available XR devices is constant during the entire execution of the application.

#### Active XR device

The application always has exactly one active XR device. The initially active XR device is determined at application startup, after determining the list of available XR devices. The active XR device can change througout application execution however.

### UI mode

Supported UI modes are:

* Screen-space UI
* World-space UI

While a head-mounted XR device is active (eg Oculus, Cardboard, …) the only available UI mode is **World-space**.

The application has notion of an active UI mode. There is always exactly one available UI mode set as the active UI mode. The initial active UI mode upon starting the application depends on the active **XR Device** at that moment:

* Headmounted XR device 🡪 World-space UI mode
* Non-headmounted XR device 🡪 Screen-space UI mode

## ApplicationStateHome

This is the initial appliction state in which the application starts up. In this state the user can:

* Exit the application
* Enter ‘Play’ application state for a given project
* Set the application settings

## ApplicationStatePlay

In this state, the user can view a given project in one of the following manners:

* Regular mode
* VR mode
* AR mode

# ApplicationSettings

## Functionality

The **ApplicationSettings** script manages the application settings for the ArchiVR application. Upon application shutdown, the **ApplicationSettings** script stores the currently active settings to the presistent file **applicationsettings.data** in the Application’s persistent data folder. Upon application startup, the **ApplicationSettings** script loads the application settings from this file. The application settings are composed of **GraphicsSettings** and **ControlsSettings**.

## GraphicsSettings

The **GraphicsSettings** are composed of:

* Quality Level (String)
* Dynamic Vegetation (bool, default: on)
* Fog (bool, default: on)

### Quality Level

#### Functionality

The quality level defines the quality of graphical rendering of the application. For each platform a default quality level is defined in the Unity editor. The quality level can be changed at runtime however.

Quality levels vary from very high quality (ultimate) to very low (very low). Every quality level implies a certain tradeof between graphical quality and performance. The higher the quality levels can only be run on high-power systems, while the lower-quality levels can still run smoothly on lower-powered systems, albeit at the expense of a decrease in graphical quality.

## ControlsSettings

**ControlsSettings** define how the user interacts with the application.

**ControlsSettings** are composed of:

* NavigationMode (One of: FPS, Fly, TelePort, Tracked, VuforiaAR)
* TrackingSystem
  + WMTracker
    - Enabled (default: false)
    - IP
    - Port
  + WindowsXRTracker
    - Enabled (default: false)
    - ???

### Navigation mode

The application has notion of following navigation modes:

* FPS
* Fly
* TelePort
* Tracked
* VuforiaAR

#### Input Mode

Navigation modes FPS, Fly and Teleport all support following input modes:

* TranslationInput
  + Mouse
  + Keyboard
  + Gamepad
  + Touch (Virtual gamepad, Tap, Swipe)
  + GearVRController
* RotationInput
  + Mouse
  + Keyboard
  + Gyro
  + Gamepad
  + GearVR Touchpad
  + Touch (Virtual gamepad, Tap, Swipe)
  + XXXTracker
  + GearVRController

By default, when the application starts up for the first time, or after a reset of the application setting,

The application starts up in a VR-style mode if supported:

* GearVR
  + Default rotation control: FromGyro
  + Default translation control
    - Gamepad
    - GearVRGamepad
    - GearVRTouchpad
* Tracking
  + MicrosoftXRTracker
    - Default rotation control: FromGyro
    - Default translation control
      * Tracking
      * Gamepad
  + WMTracker
    - Default rotation control:
      * FromGyro
        + option ‘used tracking for absolute correctness’
    - Default translation control
      * Tracking
      * Gamepad

All suitable and supported Input modes are enabled

## DebugSettings

The DebugSettings is composed of:

* Debug level

The DebugSettings option is only visible when debugging is enabled.

## How to use the ApplicationSetting

Every main scene in the Application needs to have an active ‘empty’ root gameobject with the ‘ApplicationSettings’ script attached to it.

The Applicationsettings is a singleton. It is also persistent during the execution of the Application: Once it is created, it sets itself as ‘DoNotDestroyOnLoad’, which makes that this instance stays alive when transitioning between scenes.

# Virtual gamepad

The virtual gamepad is a non-physical gamepad that exists of touch-interactable UI controls. A virtual gamepad is only shown when the following preconditions are met:

1. The system has touch input support.
2. The option [Settings] > [Control Settings] > [Show virtual gamepad] is enabled.
3. The current navigation mode supports virtual gamepad (? Don’t they all?)

Each navigation mode can support its own gamepad implementation, containing only those controls that make sense for that navigation mode:

* FPS
  + ‘Walk’ D-pad: bi-directional D-pad to walk forward/backward (up/down) and left/right (left/right).
  + ‘Run’ button: While pressed, the player runs instead of walks.
  + ‘Jump’ button: When pressed, the playe jumps
* Fly
  + ‘Move Forward/Backward/Left/Right’ D-pad: bi-directional D-pad to move forward/backward (up/down) and left/right (left/right).
  + ‘Up/Down Translation’ D-pad: vertical-only D-pad to increase/decrease the vertical position of the player in the virtual world.
* AR
  + ‘Vertical Translation Offset’ D-pad: vertical-only D-pad to increase/decrease the vertical offset of the virtual model from the physical marker’s anchor position.
  + ‘Rotation Offset’ D-pad: horizontal-only D-Pad to adjust the rotational offset of the virtual model irt the physical marker’s anchor position.
* Teleport
  + Button ‘Teleport’: Makes the player teleport tot he currently gazed-at position in the virtual world.

All gamepads are located under a root ‘VirtualGamepad’ gameobject. This root GameObject has a ‘VirtualGamepadManager’ script attached. The VirtualGamepadManager’ script is a singleton, that manages all virtual gamepads in the application.

# Project

## Prerequisite Unity scene layout:

* World
  + Phases
    - Begin
      * WorldBounds
      * Construction
        + MeshImportedFromSketchup
      * Furniture
        + Layer\_Terrain
        + Layer\_FloorL0
        + FloorL1
    - Final
      * WorldBounds
      * Construction
        + Mesh
      * Furniture
        + Layer\_Terrain
        + FloorL0
        + FloorL1
* Application
* UIManager
* …

## ProjectLayers

The scene content can be arranged into several project layers. A project layer contains related scene content that can be hidden/shown as a whole.

### Layer menu

The ‘Layer’ menu allows to set the visibility of project layers. The ‘Layer’ menu is only accessible while in Play state. The Layer menu can be opened from the main menu using the ‘Layer Menu’ button. The ‘Layer’ menu consists of the following components:

* An ‘Exit’ pushbutton.
* A ‘Show All’ pushbutton.
* A vertical scroll view containing a Layer option UI control for each loaded layer.

A Layer option UI control consists of:

* a checkbox button
  + shows the visibility of the corresponding project layer, and also can be used to set the visibility of that project layer.
* a text caption.
  + shows the name of the corresponding project layer.

TODO:

Refine to accomodate for large sets of layers simultaneously:

1. Add a vertical scroll view to contain layer buttons.
2. Make scroll view resize when adding layer option UI controls.
3. Make scroll view show scroll bar.
4. Make scroll bar useabe using different input methods:
   1. Mouse
   2. Keyboard
   3. Touch
   4. Gaze input
   5. GamePad
   6. GearVR trackpad

### Layer Manager

Project layers are managed by a **LayerManager** singleton instance. The LayerManager manages a list of uniquely named project layers. It offers the following functionality:

* Compose the list of project layers from a project, upon loading that project.
* Get the list of layers in the currently loaded project.
* Get a layer by name.
* Set the visibility of all layers.
* …

#### Adding a GameObject to a layer

To add a GameObject to a layer in the unity editor, name it in accordance with this naming convention:

|  |
| --- |
| “Layer\_” + insertLayerNameHere |

SomeExample names for GameObjects that belong to a specific layer in a typical layer setup:

* Layer\_Terrain
* Layer\_Facade
* Layer\_Furniture
* Layer\_FloorB1
* Layer\_FloorL0
* Layer\_FloorL1
* Layer\_FloorL2
* ‘Layers’ menu

## POI

Points-of-interest are predefined reference positions from which the architectural project can be viewed. The points of interest are defined in the unity editor (see [Procedures] > [Add construction project] > Define POI.

# Layers

## VREyeRayCastIgnore

This layer contains all scene objects that have a collider, but should be ignored by the **VREyeRayCast** component on the main **Camera**, because they are not a **VRInteractiveItem**.  
Contains:

* Panel\_MenuFront\_VR
  + Has a collider in order to detect whether the user is looking at the VR front-facing menu. If this is the case, the menu does not adjust its rotation. Else it updates its roatation continuously, in order to always be located in front of the main Camera (~stay in front of the player.). Panel\_MenuFront\_VR must be excluded because it works in tandem wit hits own RayCaster, and not VREyeRayCaster, and would otherwise block VRInteractiveItems in the scene, like eg in the Home state, VR Buttons to open a project.

## Layer-based collision

The GameObjects for world-space UI controls are part of a separate Layer ‘UI’. The collision matrix fort he ArchiVR project is setup in such a way that colliders from the default layer do not collide with objects from the UI Layer. This is done to prevent the FPS collider from colliding with eg the world-space floor menu and its components.

# RenderLayers

The scene content is arranged into several Render layers:

1. Sky
2. Celestial objects
3. Horizon
4. World
5. UI-VR
6. UI

The layers are rendered in the order in which they are listed above. This is achieved by managing the layers in Unity (via TODO > TODO >)

The layers are managed in Unity editor. A GameObject can be assigned to a Render Layer by setting the ‘Layer’ field of the Gameobject in the ‘inspector’ pane.

## Layer ‘Sky’

Contains the skydome.

Clears both dept hand color buffer (with a solid color).

## Layer ‘Celestial objects’

Contains the celestial objects like sun and moon.

Only clears depth buffer.

## Layer ‘Horizon’

Contains the horizon dome.

Only clears depth buffer.

## Layer ‘World’

Contains the virtual environment and objects in it.

* Terrain
* Buildings
* Furniture
* ? World-space interactive items
  + Light switches
  + …
* …

Only clears depth buffer.

## Layer ‘UI-VR’

Contains the world-space UI components that are attached to the player for VR mode.

Only clears depth buffer.

## Layer ‘UI’

Contains the screen-space UI components for non-VR mode.

Only clears depth buffer.

# Procedures

## Procedure: Add a construction project

1. Import assets
   1. model of building (Sketchup v2015), into Assets/KS/Model/ProjectXXX folder.
   2. Project preview image, as 2D/Sprite, into Assets/Resources/ProjectPreview/ folder.
2. Create project scene
   1. Named ‘ProjectXXX’, in folder ‘Assets/Scenes’
   2. Hint: Copy-Rename-Modify an existing, functional project scene.
3. Add Lighting to project scene
   1. World/Lighting
      1. LightgroupXXX
      2. Light01
      3. Light02
      4. …
4. Add POI to project scene
   1. POI.Default
      1. L0Leefruimte
      2. …
5. Add project selection button to the Home state’s Project Selection menu
   1. This could be automated, so project selectionn scene buttons are dynamically creadted and arranged according tot he project scenes available.

### Define dynamic vegetation

### Define construction lighting

Under the ‘Construction Lighting’ Gameobject below the concened ‘Construction Phase’ GameObject, create a hierarchy of GameObjects that is equal tot he Layer hierarchy, eg:

* Lighting
  + B0
    - Stair
    - Bicyle Room
    - Main Storage
    - Technical Room
    - …
  + L0
    - Entrance
    - Kitchen
    - Living Room
    - Dining Room
    - Toilet
    - Stair
    - …
  + L1
    - BathRoom
    - Toilet
    - Bedroom 1
    - …

Then add a GameObject for each LightGroup into the corresponding room’s GameObject, eg:

* Lighting
  + LO
    - Living Room
      * Lightgroup Plafond
      * Lightgroup Wall
      * Lightgroup Standing lamp

Then add lights to the lightgroup GameObjects. A collection of light fixture prefabs (wit hand without corresponding Unity Light object) are readily available in the assets folder ‘Models’.

### Define POI

Points-Of-Interest are defined using the Unity editor. First a ‘POI collection has to be defined:

Create an empty GameObject under [ProjectPhase] > [POI].  
The POI collection has to have the name of the navigation mode in which the POI can be used.  
Example:

* Phase Final
  + POI
    - FPS
      * Slaapkamer
    - Fly
      * Slaapkamer
      * Vooraanzicht VL
    - Teleport
      * Slaapkamer
    - AR
      * Slaapkamer
      * …

Then new POI must be added tot he POI collection as follows:

1. Add a new ‘Camera’ beneath the ‘POI collection’ GameObject (eg POI.default)
2. Rename the POI to a descriptive name (eg ‘Kitchen’, ‘Bedroom 1’, …)
3. Remove the ‘SoundListener’ component from the camera
4. Put the POI in the correct location:
   1. Translate and rotate it into the correct location, while reviewing the POI resulting view in the Unity editor ‘Game’ window.
   2. To make the Unity editor show the POI view result, have only the concerned POI’s Camera component active – disable all other cameras in the scene.
5. When the POI is correctly located, make sure the POI GameObject is enabled, but disable its ‘Camera’ component.

## Procedure: Remove a construction project

Remove the project scene

Remove the project preview Sprite asset.

Remove the project selection button from Home state’s Project Selection menu.

# Development tools

## Monitoring application performance

The FPS can be investigated using the FPS counter widget and exported file.

## Developing/debugging a project scene

When making a lot of successive changes to a project scene, the easiest way to review and debug those changes is to set the ‘InitialProjectName’ setting on the PlayApplicationState. This makes the Play state start up with the designated scene loaded right away.

# TODO:

## Code implementation

### When switching to Camera Navigation mode Fly, properly set camera position to a POI

### Fix toggle UI using tap

Toggling UI on/off with tap not on UI control is broken.

### Properly implement ‘Controls’ menu.

For now the ‘Controls’ menu is part of the Debug tab view (tab 0). It should be a separate menu instead, with ‘Exit’ button etc…

### Implement infrastructure for user interaction in menu using different input devices

* Mouse (done)
* Gaze (done)
* Keyboard
* Gamepad
* GearVR Trackpad

### Implement user interaction in menus using keyboard

* **Esc**: Cancel / Exit menu
* **Enter**: Confirm
* **Arrows**: Navigate up/down, left/right
* **Tab:** Focus first/next element
* **Shif+Tab:** Focus last/previous element

### Implement user interaction in menus using gamepad

* **A**: Cancel / Exit menu
* **X**: Confirm
* **Analog joysticks/ Discrete D-pad**: Navigate up/down, left/right
* **R1:** Focus first/next element
* **R2:** Focus last/previous element

### FPS navigation: Fix walk speed when using keyboard.

Investigate influence of ‘Mobile Input’ option in Unity editor:

ON: Walk speed is too high, looks like always running.  
OFF: Walk by default, run while shift pressed.

### Implement option ‘Gyro’ in ‘Settings>Controls’ menu.

Implement generating screenshots using a specified resolution.

Allow the generation of screenshots to a resolution independent of the application resolution Example: full-HD (1920\*1080)

Implement generating screenshots using best available graphics quality level.

Activate best available graphics quality setting when generating screenshots.

### POI

#### Properly implement and document a way to generate POI.

* Shortcut key = ?
* To File (path:???)

### Sky

#### Dynamic cloud layer

Implement a dynamic sky layer based on unity ‘dust field’ prefab.

* Based on perlin noise?
* Add heightmap for extra visual ‘depth’

#### SkyDome

* Implement Finegrained control over Sky gradient.
  + Shader: add property \_SkyLight1InfluenceRangeAngle
* SkyDome Shader: implement properly support for arbitrary number of celestial objects.
* Sky behavior: use SkyDome Shader support for arbitrary number of celestial objects, to properly represent both sun and moon.
* SkyDome : Implement making rendering ground optional.
* SkyDome : Implement timed ‘Ground Colors’
* Implement Fog ?

## Project scene

### Project 006

#### Create ‘Furniture-less’ version of sketchup model (export as sketchup version 2015)

#### Add Furniture from Assets Model folder

#### Add terrains with dynamic vegetation

* Terrain ‘Tuin Achter’
* Terrain ‘Tuin voor’
* Add vegetation (hedges/trees) as ‘mesh detail’ tot he terrains.

### Project 008

#### Create ‘Furniture-less’ version of sketchup model (export as sketchup version 2015)

#### Add Furniture from Assets Model folder

#### Add proper lighting

### Project 011

#### Create ‘Furniture-less’ version of sketchup model (export as sketchup version 2015)

#### Add Furniture from Assets Model folder

#### Add proper lighting

#### Add proper furniture

### Project 024

#### Create ‘Furniture-less’ version of sketchup model (export as sketchup version 2015)

#### Add Furniture from Assets Model folder

#### Add proper lighting

### Project 025

#### Create ‘Furniture-less’ version of sketchup model (export as sketchup version 2015)

#### Add Furniture from Assets Model folder

#### Add proper lighting

## Prefabs

### Lighting

#### Review and finalize reusable lighting components as prefabs

### Furniture

#### Prepare a ‘Model’ folder with reusable ‘Furniture’ components as prefabs.

For each reusable model:

* In sketchup
  + Review model
    - Faults (inverted/missing faces, uv mapping)
    - Correct units? (must be mm)
    - Correct scale?
  + Save copy as as sketchup v2015 .skp file.
* In Unity editor
  + Prepare a folder fort he model
  + Load skp model file
  + Set material settings.