AMBIENT ASSISTED LIVING JOINT PROGRAME

AAL-2013-6-039

SeniorLudens

Serious Games development platform for older workforce training and intergenerational knowledge transference

D3.5
SeniorLudens serious game engine and platform technical evaluation report (1)

Workpackage	WP3 - SeniorLudens platform design and implementation
Lead beneficiary	INDRA
Editor(s)	Dani Tost- CREB-UPC Ariel von Barnekow – CREB-UPC Núria Bonet Codina – CREB-UPC Salvador Aguilar - INDRA
Contributor(s)	Stefano Puricelli - CBIM
Reviewer(s)	Stefano Puricelli - CBIM Salvador Aguilar - INDRA
Release Date	03/2015
Version	V1.0
Circulation	Project Partners, AAL Control Management Unit, and National Funding Agencies.

Table of Contents

Abstract

This document comprises the technical definition and evaluation of the major elements in SeniorLudens System: SeniorLudens Game Engine and SeniorLudens Platform.

SeniorLudens Game Engine is detailed in the first section, which is composed of two parts: in the first part it describes the SeniorLudens Serious Game Engine composed by the Game Kit (SLGK), Task Editor, Scenario Editor and the Training Program Editor. It defines the current features of the software, its structure and the tools used for its development. This part completes the definition of the game engine presented in Deliverable D3.1 and the user manual that comes with the demonstrator D2.5. The second part of the deliverable is devoted to the evaluation of the platform, focusing on the component which will not be covered by D4.2 the game kit. The evaluation has been done after a questionnaire for developers of new worlds on top of SLGK have answered about the capabilities, usability and development speed.

SeniorLudens Platform is covered in the second section, following the same two parts utilized in the previous section. The first part details the overall design of the present status of the development. The second part intends to detail the current evaluation of the system according with the technical requirements defined at the beginning of the project.

1- SeniorLudens Game Engine

SeniorLudens Game Engine is formed by four of the components of the SeniorLudens platform, the training program editor, the task editor, the scenario editor and the game kit.

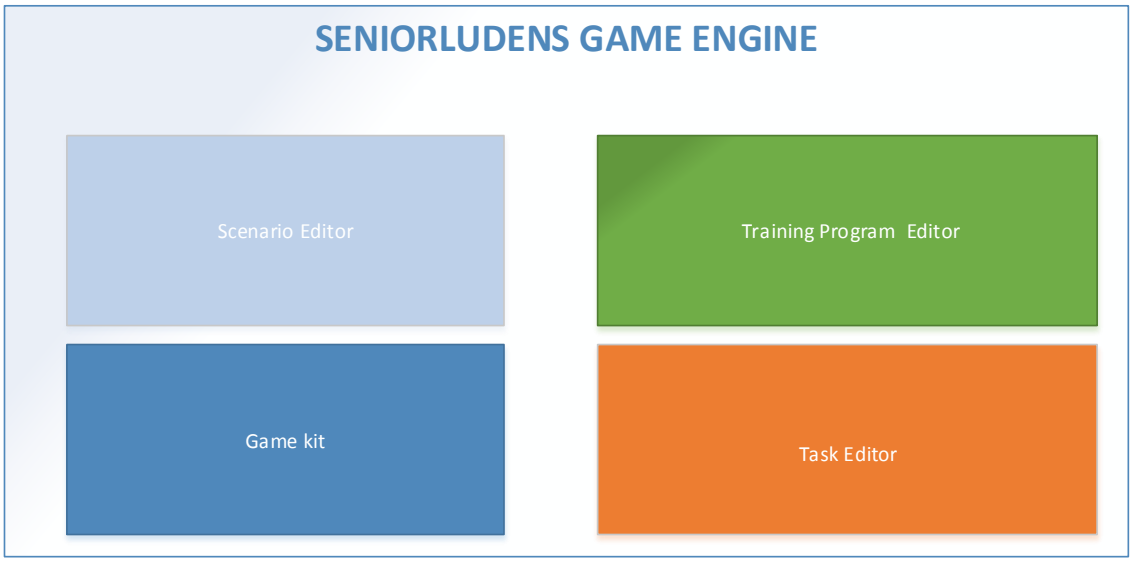

Figure 1- SeniorLudens Game Engine Components

This document covers the three game engine components developed during this phase of the project.

1.1- Game Kit

The SeniorLudens Game Kit (from now on SLGK) provides an abstract definition of the virtual environments and the training activities that can be performed in them, and it implements these definitions on top of a game engine in order to allow the creation of training game. Currently, the SLGK has an interface to Unity3D, but it could be extended to other Game Engines.

The aim of SLGK is to provide enough functions to design and implement serious games for training. The objects and actions provided by the toolkit will give support to all the requirements of the three use-cases: traditional cheese creation, rehabilitation planning training and IT project planning and development. These three use-cases represent a wide range of objects and situations; therefore, SLGK will allow to use these functionalities to create new games in new virtual worlds with a large variety of actions and behaviours, without need to program new features. However, SLGK is opened to add new categories of objects and new actions.

In terms of simulation of real-world situations, SLGK provides the following features:

- Type of environment and perspective
  - 3D indoor environments (see Figure 2): with navigation restricted to relatively small paths, navigation of a first-person perspective at the human height (human-walk view). In these environments, the perspective on the objects is near, thus, on one hand, objects should have a more accurate representation and, on the other hand, user interaction can be more precise. The collision of the user-avatar tied to the camera with the scenario should be controlled in order to avoid the user penetrations into walls and furniture.

Figure 2 - Example on an indoor SeniorLudens scene. This scene was designed during the first phases of development, as a proof-of-concept of the SLGK.

  - 3D outdoor environments (see Figure 3 - Example of an outdoor environment. In this case it is a scene of the case study GrowYourProject.). In these environments the camera perspective can be set to isometric view, with the camera floating far from the scene on an elevated plane (air-plane view). In these environments, the number of objects can be higher, because the landscape is larger, but the level of details needed for their representation can lower that in the former case. In general, the collisions of the user avatar with the scenario are avoided, because there are no objects in the camera plane. In these environments, even much more than in the closed environments, complementary tools of camera handling are necessary in order, for instance, to zoom-in and zoom-out or change the view from air-plane to ground view. These

actions are not provided in the current version of the SLGK, but they are planned for the next release.

Figure 3 - Example of an outdoor environment. In this case it is a scene of the case study GrowYourProject.

- Automatic navigation: a click on an object drives the virtual avatar and the associated camera near enough to the object to include it within the selection range.

- Selection by click on an object: a click on an object near enough to the camera results on a selection of the object. If the click is done through a mouse o keyboard device, the selection allows to distinguish which kind of selection has been done (primary, secondary or, eventually, higher order)

- Basic and specialized objects behaviours.

- Interfaces (see Figure 4):

  Every world can have one or more interfaces available composed of messages boxes and option menus. The actions currently available on these interfaces are the following:

  - Show a specific interface item
  - Hide a specific interface item
  - Modify the text contents a specific interface item
  - Show a specific panel of the interface
  - Hide a specific panel of the interface

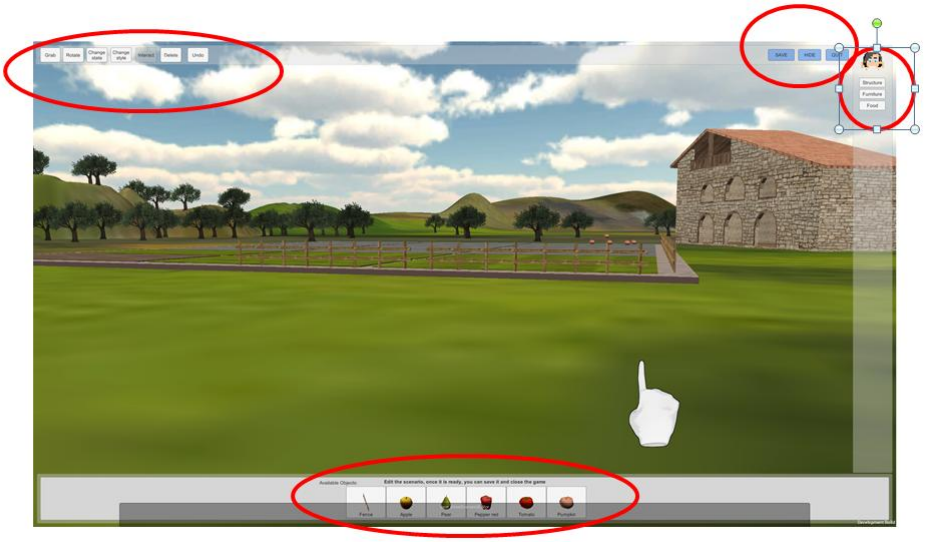

Figure 4 - An example of an interface object provided by SLGK. It is the preliminary interface used in the Scenario Editor Game. The interface is composed of four panels (surrounded in red). SLGK provides actions to show and hide them and modify the text of the buttons. The interface also contains a message box currently hidden by the objects menu at the bottom part of the screen.

1.1.1- Game-loop structure and implementation

1.1.1.1- Main components

As explained in Deliverable 3.1, games occur in *Scenes* of *Worlds*. The set of scenes of a game forms a *Scenario*. The initial setting of each scene is called *Scene Configuration*. A scene configuration defines the position, orientation, state and visual style of every object instance present in the scene. Throughout the game, as a consequence of user actions, time-dependent events or system actions, the state, position, orientation of these object instances can change. Some of them can disappear and other be created. Each game has a narrative thread, composed of three stages (introduction, development and conclusion) each divided into different tracks that define the expected behaviour of the trainee, i.e. the actions that he/she is expected to launch, the autonomous behaviour of the objects and the trainer actions.

SLGK games is thus structured around three basic elements (see Figure 5: The three SeniorLudens games main elements: the World, the Scene Configuration built from the Scenario description file and the Reference Task built from the task description file.):

- The definition of the *World* embedded in the code,
- The *configuration of the scene* built from an xml file (*Scenario Description File*) created in the *Scenario Editor*
- The *Reference Task* built from an xml file (*Task Description File*) created in the *Task Editor*.

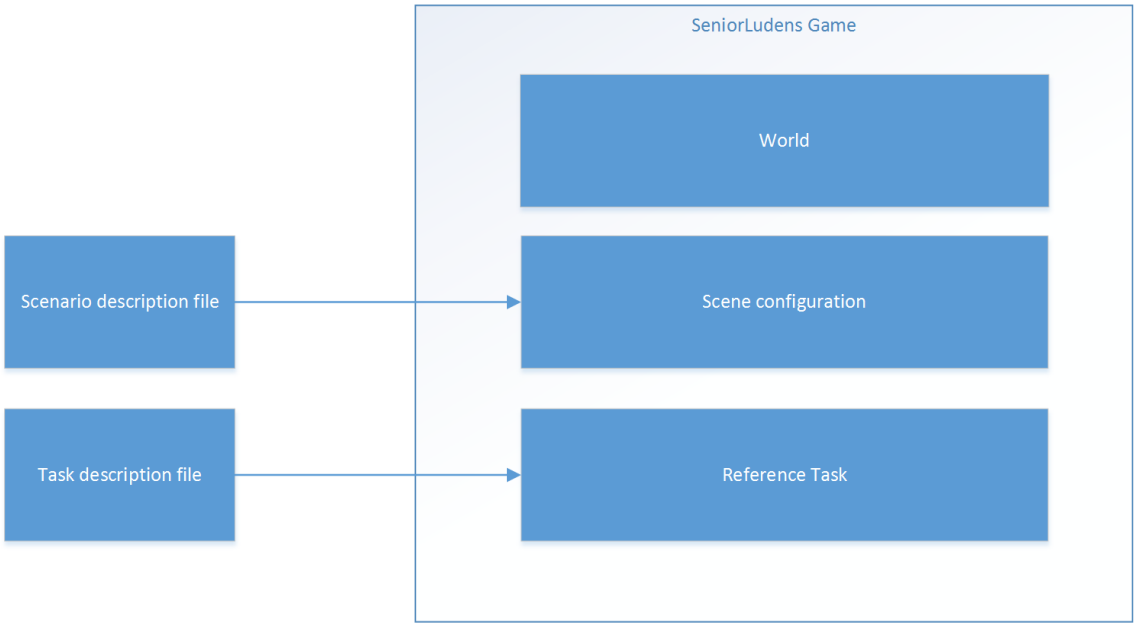

Figure 5: The three SeniorLudens games main elements: the World, the Scene Configuration built from the Scenario description file and the Reference Task built from the task description file.

1.1.1.2- Game loop actions

In run-time, a game is structured on top of a loop (game-loop) that processes user-events, executes system actions including action driven by the virtual trainer character and object's actions, computes collisions and draws the scene (see **¡Error! No se encuentra el origen de la referencia.**).

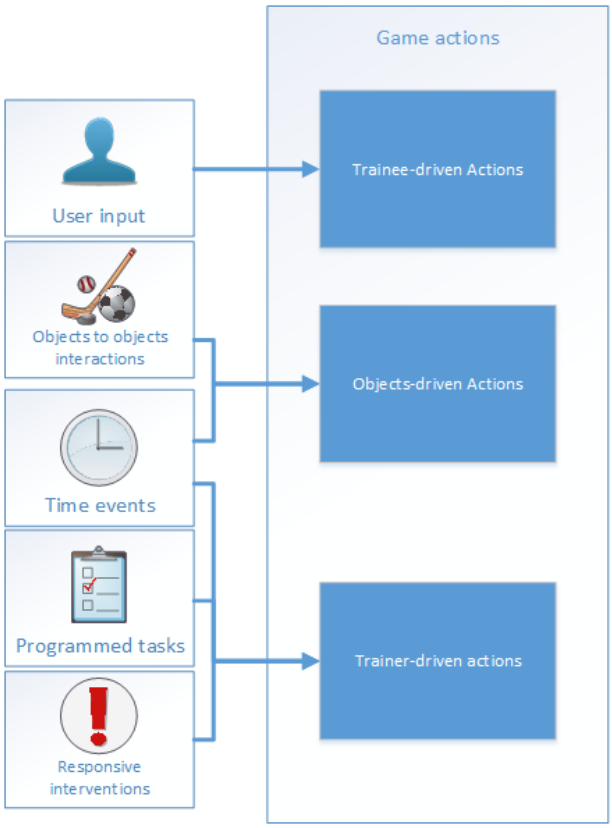

Figure 6: Actions that may occur at each loop of the game: trainee-driven actions, objects-driven action and trainer-driven action.

Specifically, trainee-driven actions are launched through user-interactions on visible objects. Every user interaction is analysed and eventually mapped to a trainee-driven action that is realized according to the current permissions regulated by levels of difficulty. Objects-driven actions occur because objects have an autonomous behaviour regulated by a time-clock (for instance, animations), or as responsive behaviour to interactions with other objects. For instance, collisions between objects yield objects movement. Some objects are connected on to the other and thus a change in one object yields to changes on the other. For instance, a thermometer changes its temperature attribute according to the object that it monitories. Finally, trainer-driven events are system-actions that represent the behaviour of the trainer. They can be programmed tasks, either depending from time or from the state of the scene or the scoring of the game, or they can be responsive actions to user actions or omissions (messages, corrections…etc.).

1.1.1.3- SLGK and Game Engine processing levels

The SLGK is composed of two layers: the abstract layer, independent from the visual implementation and the interface layer that in charge of the communication with the game engine (Unity3D). The implementation of the game-loop occurs at two different levels: at the game-engine (Unity3D) level and the SLGK level. Some actions are done at the abstract level and others at the visual level. The intermediate layer of the SLGK is in charge of the communication between the abstract layer and the game engine. Figure 7 shows schematically the level of the different actions.

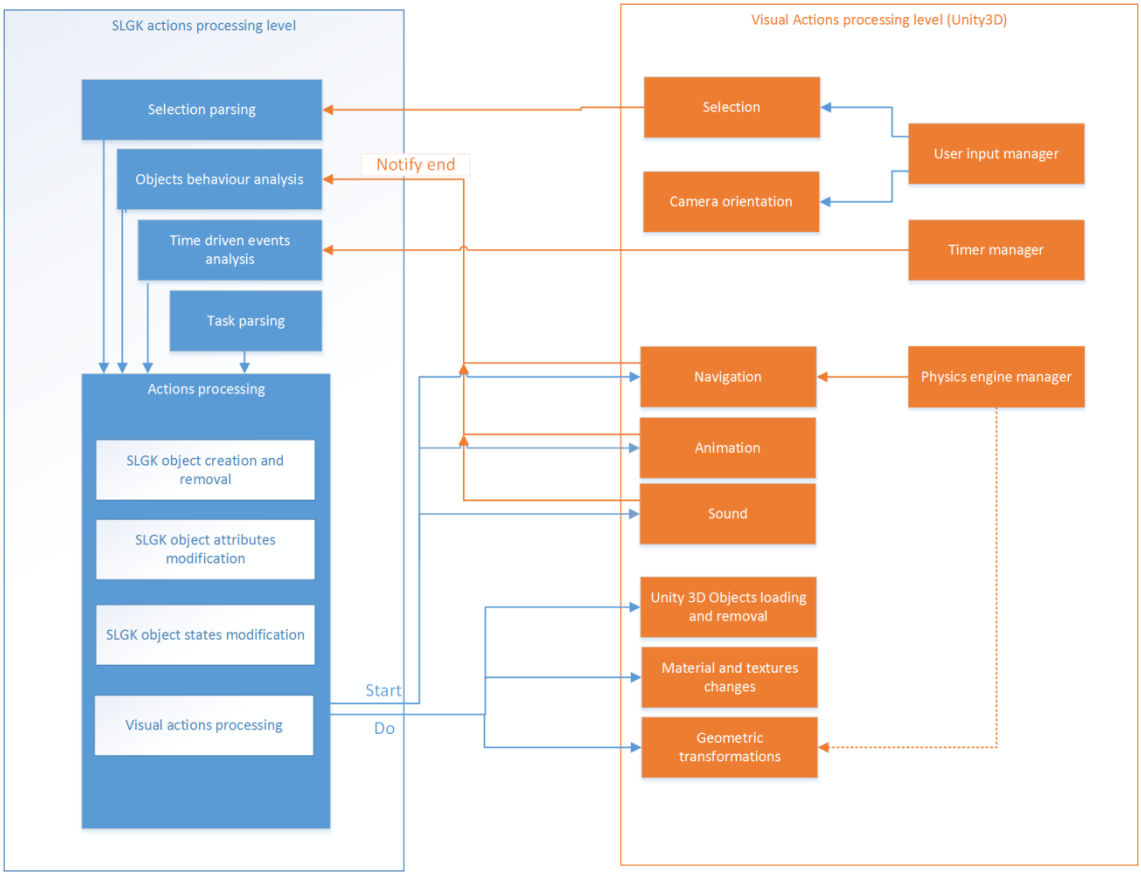

Figure 7: The two-levels of actions processing in SLGK. In blue the abstract layer and in orange the game-engine level.

Specifically, the game engine is responsible of collecting user input: mouse movement and button clicks, keyboard keys or screen touch in touch devices. The current version of SLGK is restricted to mouse and keyboard events, but in the next release its extension to touch devices is foreseen. User clicks are classified into primary or secondary events depending on if they are done with the LMB (currently set to primary) or (RMB) (currently set to secondary). Higher order actions can be associated to other types of input. Primary, secondary and higher order input categories allow to discriminate between possible actions available on an object.

User events can be classified into two main types:

- Selection: currently mouse-click on an object
- Camera orientation: currently mouse movement

The events that are recognized as camera orientation are directly held at the Unity3D level without passing to the abstract layer. Selection events are notified to the SLGK: selected object and action level (primary/secondary). The SLGK Selection Parser determines which action is being requested by the trainee and eventually launches the action.

Unity3D also manages timers and informs the SLGK. Taking into account time, the objects behaviour and the reference task, the SLGK also programs objects actions and trainer actions as explained in Section 1.1.1.2-.

The action processing is done at the SLGK engine level that modifies the world objects accordingly and launches visual actions at the game-engine level (Unity3D). Navigation, animation and sounds are actions that last through time. Thus, for these actions, and for efficiency reasons, the SLGK only requests their start. Then, during the entire time interval that

they last, it is the Game Engine that updates them without need of passing through the SLGK level. The game engine notifies SLGK that they have ended. Navigation uses the Unity3D physics engine in order to detect and prevent collisions. This avoids that the user avatar could penetrate into solid objects. Currently, the geometrical transformation of the objects (grab, rotate and scale) is not connected to the physics engine. Thus, inter-objects penetration could happen in the game. Avoiding them is a feature foreseen for the next version of the SLGK.

1.1.2- User-driven actions parsing

Figure 8: Processing of a trainee-requested action shows the process of User Input to Action Processing. As exposed in Figure 7, Unity3D performs the selection and notifies to SLGK which object has been selected.

From the selection and using information on the object and of the current state of the user avatar, SLGK computes the list of actions that match with the current information. From the current state of the object and its states graph, only the actions that are out-edges of the state in the graph are analysed. From them, depending on the grammatical structure of the action and on if the trainee avatar holds or not an instrument, only grammatically valid actions are selected. In order to select one of these possible actions, the reference task that contains a description of the expected behaviour of the trainee, allows giving priority to the most probable ones. A priority list is used to take the final decision.

Once the user requested action has been computed it is authorized or not, depending on the current level of difficulty and the Reference Task. If the action is correct according to the Reference Task, it is sent to the action processor. Otherwise, it can be sent or not depending on if the level of difficulty allows to do incorrect or unnecessary actions. The current version of the SLGK considers only two levels of authorization: all actions allowed, only actions related to the current step of the Reference task.

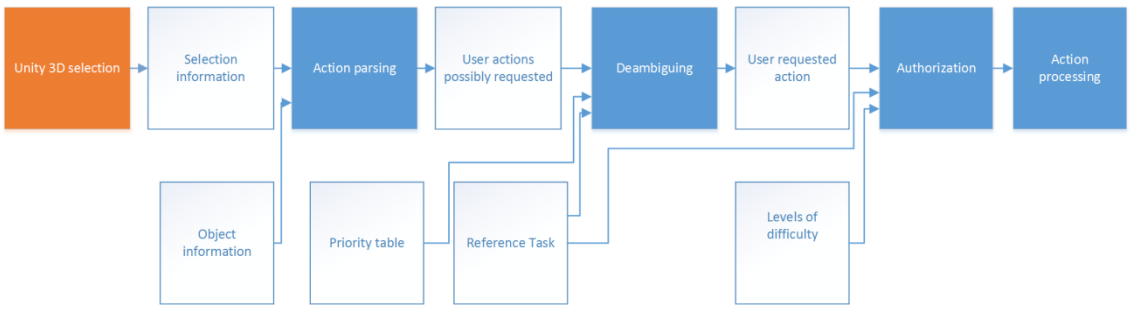

Figure 8: Processing of a trainee-requested action

1.1.3- Reference Task Tracking

The Reference Task is implemented following the definition done in Deliverable D3.1: as a set of three stages (introduction, development and conclusion) each composed by independent tracks.

A track is a composition of actions to be done either by the trainee or by the trainer or by the objects autonomously. Currently, SLGK implements three types of compositions:

- Sequential blocks of blocks:
  `Bs = Seq[B_1, B_2, B_3, …, B_n]` such that `Bs` is done if first `B_1` is done, then `B_2` and so on until `B_n`.

- Parallel blocks of blocks:
  `Bp= Set_m([B_1, B_2, B_3,…, B_n])`, meaning that `Bp` is done if `m` of the `n` blocks of the set is done, no matter in which order.

- Conditional blocks of blocks:
  Bc = (Boolean expression, B), meaning do B when the Boolean expression is fulfilled. Currently, the Boolean expression can be defined only in term of:
  - Object attribute value property
  - Time value

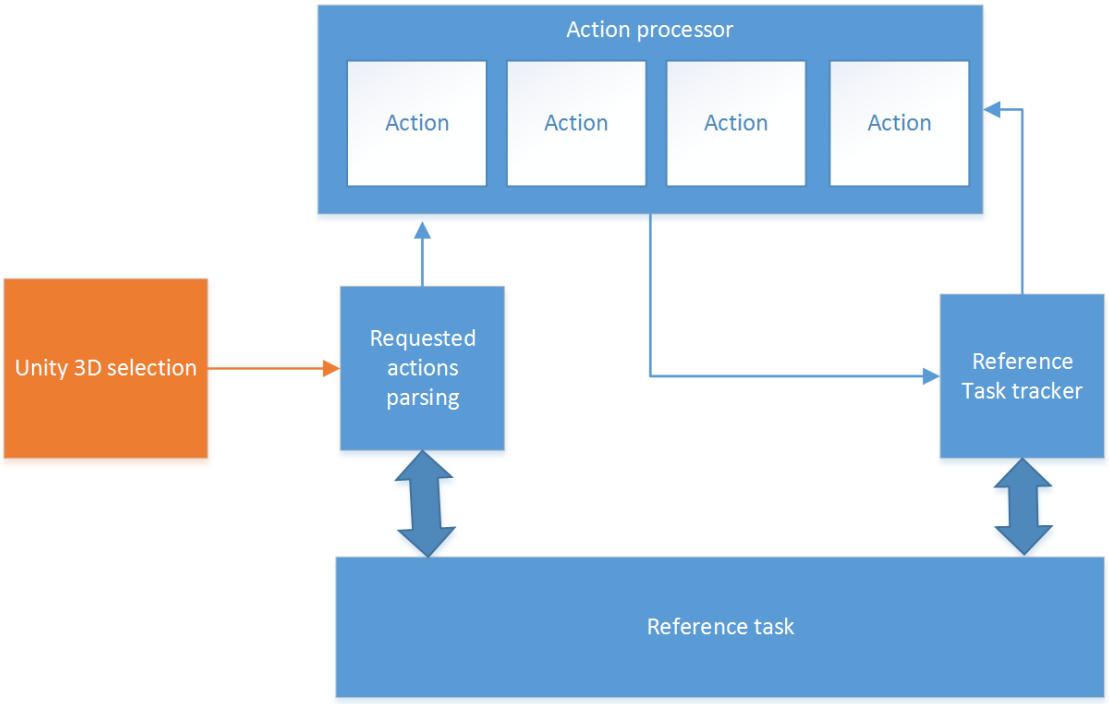

Figure 9: Tracking of the reference task

During the game, the Reference Task Tracker tracks the trainee actions by comparing them to the expected actions of the Reference Task. The action processor notifies the tracker of the actions that have started and those that have ended. This way, the tracker follows the foreseen storyboard of the game. It is able to program responsive trainer actions and to determine when a step of the game has finished. This structure makes it possible to evaluate the trainee actions. In the current version of the SLGK the tracker is implemented but not the validation in top of it.

1.1.4- Objects

SLGK provides two different ways to create objects, with visual representation and without, for the first ones the Unity3D binding integrates the Unity3D objects with game kit.

The SLGK provides two different behaviours of objects one for characters and another for objects, however if the new object requires a special behaviour the game kit can be extended to include it.

The game kit doesn't provide objects, it provides object behaviours, and the game kit SDK, a warehouse is provided as reference.

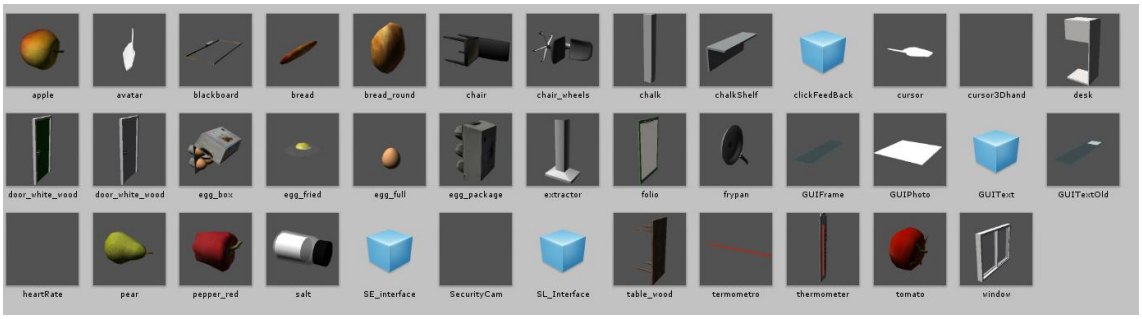

Figure 10 Some Objects provided as reference

1.1.5- Actions

When implementing an action in the SLGK three different approaches can be used: atomic actions, composite actions, or alias action.

Atomic actions are self-contained, they do not need any other action to work, normally those actions do only one thing and they have some options to customize their behaviour. The customization of the action can be done when the action is assigned to an object.

If basic actions are self-contained, *Composite Actions* are defined using other actions (basic, composite or aliases), the composition of those actions is defined either as a sequence or as a set, where:

- Sequence: are defined as processing order sorted lists of actions: `Ac = Seq[Ac_1, Ac_2, Ac_3….Ac_n]` such that in order to do `Ac`, first `Ac_1` must be done, then `Ac_2` and so on until `Ac_n`.
- Set: are defined as group of actions `Ac = Set([Ac_1, Ac_2, Ac_3….Ac_n])`, meaning that in order to do `Ac`, the `n` actions of the set will start together and it will be done once all finish.

Finally alias actions are used to rename and existing action, this happens when many different actions in the real world are implemented exactly in the same way in the virtual world. From the task point of view, they are treated as different actions, but the implementation is unique.

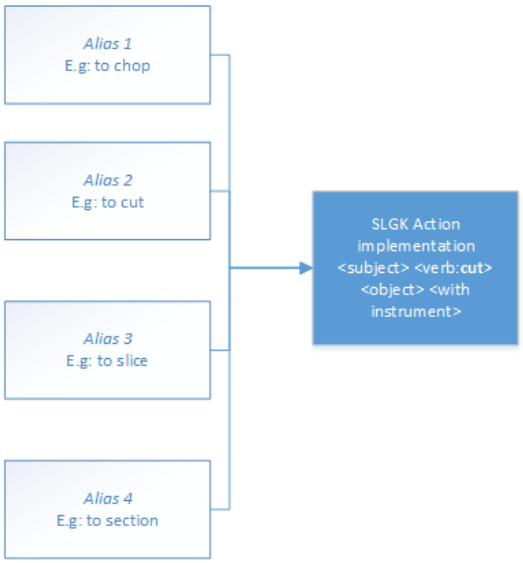

Figure 11: Mapping of different aliases actions to a unique SLGK action implementation

In some cases the level of customization of atomic action requires information not available neither in the user input or the object relation with the action, this kind of actions (basic, composite or alias) are called *System Actions* as they exist to be reused by other composite actions.

1.1.5.1- Actions provided by the game kit

1.1.5.1.1- Atomic actions

- Play animation: reproduces an animation in an object.
- Play sound: reproduces a sound
- Stop sound: stops a sound if it's playing.
- Touch: This is the most basic action, recognize with object is the clicked/touched by the user.
- Pick: an object from the scene is removed from its position and placed in the hand of the virtual avatar.
- Drop: it places the object in the hand of the virtual avatar in another object of the scene.
- Rotate: rotates an object in the scene
- Create of a new object instance: creates a new instance of an object definition available on the world.
- Remove an object instance: removes an instance from the scene.
- Change the state of an object: changes the state of the object.
- Change the material of an object: changes the material used by an instance of an object.
- Change the properties of an object: changes the value of a property of an object (system action)
- Look At: changes the orientation in the scene to look at an object or point.
- Lock Camera / Unlock Camera: moves the camera to a given point and disables the orientation, or brings the camera back to its original position and enables orientation
- Navigate: This action moves the users from their current location until they are near enough to the new destination (normally an object of the scene).
- Stop Navigation: when a navigation action is running the user may notice they didn't want to go there, this action stops the movement.
- Put On, Put At: places an object somewhere in the scene.
- Write: the keystrokes are interpreted as text or in mobile devices the virtual keyboard is throw.
- Notify: notifies an event to and object (system action)
- Quit: hides the current scene and displays if available the menu of the game.

1.1.5.1.2- Complex actions:

- Write on blackboard: changes the position of the camera and locks the orientation of the camera to improve the experience when the user is writing in the blackboard.
  Seq[Pick chalk , Lock Camera, Write]
- Create At Hand: creates a new instance of an object and places it in the virtual hand of the user.
  Seq[Create Object, Pick Object]
- Turn On/Turn Off: turns an object on or off, for example a treadmill.
  Seq[Play Animation, Notify, Change State)
- Open, close: opens/closes an object, for example a door.
  Seq[Play Animation, Change State)
- Enable Movie Mode: creates a new object in the scene that modifies the aspect of the window to inform to the user that from now on they don't have to interact with the game.
  Set(Create, Lock Camera)

- Disable Movie Mode: removes the object created by enable movie mode and restores the normal execution of the game.
  Set(Remove, Unlock Camera)

1.1.6- SeniorLudens Game Kit SDK

In order to simplify and enhance the development process of games with the SLGK, we provided a set of tools which packed with the SLGK form the SeniorLudens Gamekit SDK (from now on SDK).

The SDK is designed thinking in its extension and portability, however to improve the development workflow there is an extension of the game kit for Unity 3D (SDK for Unity3D), together they provide:

1. ***Unity3D Editor Extension:*** *this is an extension for the Unity 3D editor. It displays specific information for the game kit objects in the Inspectors and custom UI to interact with the SDK main features. It also provides a validator for the scene which detects possible problems on the current scene like missing object information, duplicate object identifier.*

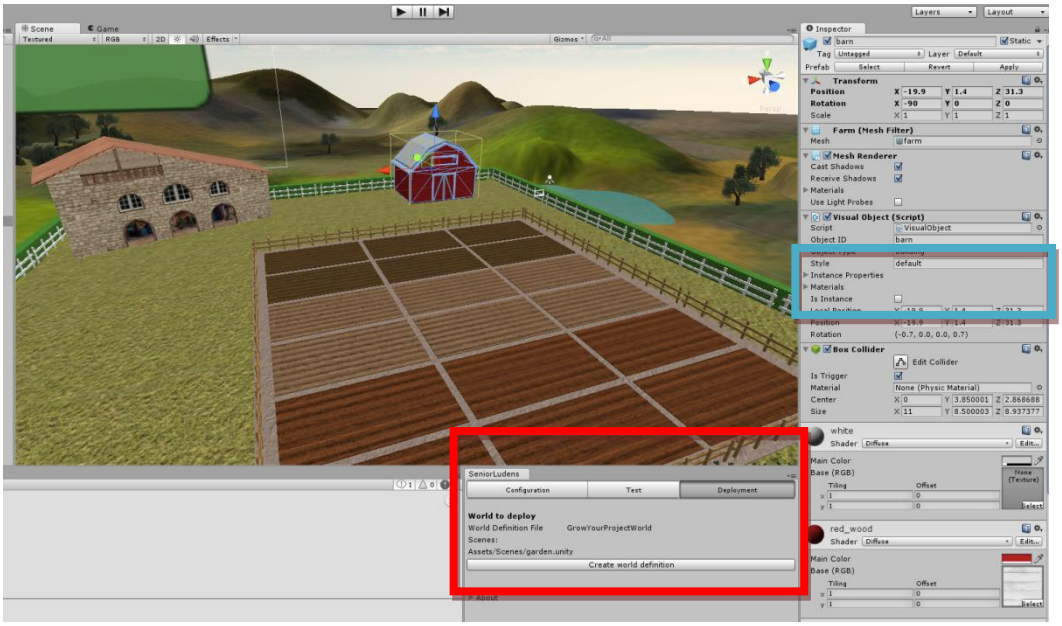

Figure 12 Details from the Unity 3D Editor Extension (in red the interface to the SDK tools, in blue custom inspector information)

2. **Warehouses**: a game data files management pattern (see the following section Warehouses).
3. ***Project examples:*** the sdk comes with two complete Unity3D projects to be used as showcase and as learning tools during the development of new scenarios using the SLGK.
4. ***HTTP SDK Server:*** this server provides an interface to interact with the tools of the sdk and also it implements the calls from the SeniorLudens Platform required by the gamekit, and can be used during the development process to check the world without having to publish the game on the SeniorLudens platfrom.
5. **Validator**: the aim of this tool is to ensure the correctness of the descriptors, and the warehouses. Currently only the validation of the descriptors using the corresponding Xml Schema is provided, there is another validator bundled inside the Unity3D editor which validates the correctness of the scenes.

6. **Descriptor Generator**: offers an automatic way to generate the world descriptor xml file needed in the Task Editor from the world data structure of the game is partially implemented.
7. **User manual and API docs**: those are not tools but they are very useful resources during the development of a game, they provide a reference of how to integrate the game kit with your game and how to extend the functionalities of the SLGK designing new behaviours and actions.

1.1.6.1- Warehouses

SLGK Warehouses are organized in a very specific way in order to foster the reuse of models, materials and textures, to avoid redundancies and to provide means of creating easily catalogues of resources. There is a general warehouse (*SeniorLudens Warehouse*) with common resources, available to all users of the SeniorLudens platform, and a warehouse for each world with the private objects of that world. The structure of a warehouse is depicted in Figure 13.

There are three main folders: *objects*, *materials* and *textures*. The *Materials* and *textures* folders contain the materials and textures that are shared by various objects of the warehouse. Besides, each object can have particular materials and textures stored in their folder as explained below.

The *objects* folder is structured into categories. Currently the definition of categories and the classification of objects into these categories is free. In the SeniorLudens warehouse for instance, there are categories such as food, kitchen (for all kitchen utensils), structure (windows and doors) and furniture. Each category is substructured into subcategories, for instance food into fruits, eggs, condiments. Finally, within each subcategory, there is a separate folder for each object. When objects are composed of different parts (a door frame, a frame and a handle, for instance), each part is considered as a different object and stored in a different folder unless they act as a whole. The object's folder contains the following files:

- definition.xml: the object definition file
- If the object has a visual object:
  - subfolder *Materials*, with the materials that are specific to that object, not shared by any other.
  - subfolder *textures*, with the textures that are specific to that object, not shared by any other.
  - Graphical models of the default style, one for each state of the object that requires a different one.
  - The Unity prefab of every state (e.g.: mop.prefab)
- If the object has more than one style: - subfolder *styles*
- If the object has its own scripts (specific update or actions): - subfolder *scripts* and within it, subfolder *cs* for the C# scripts.

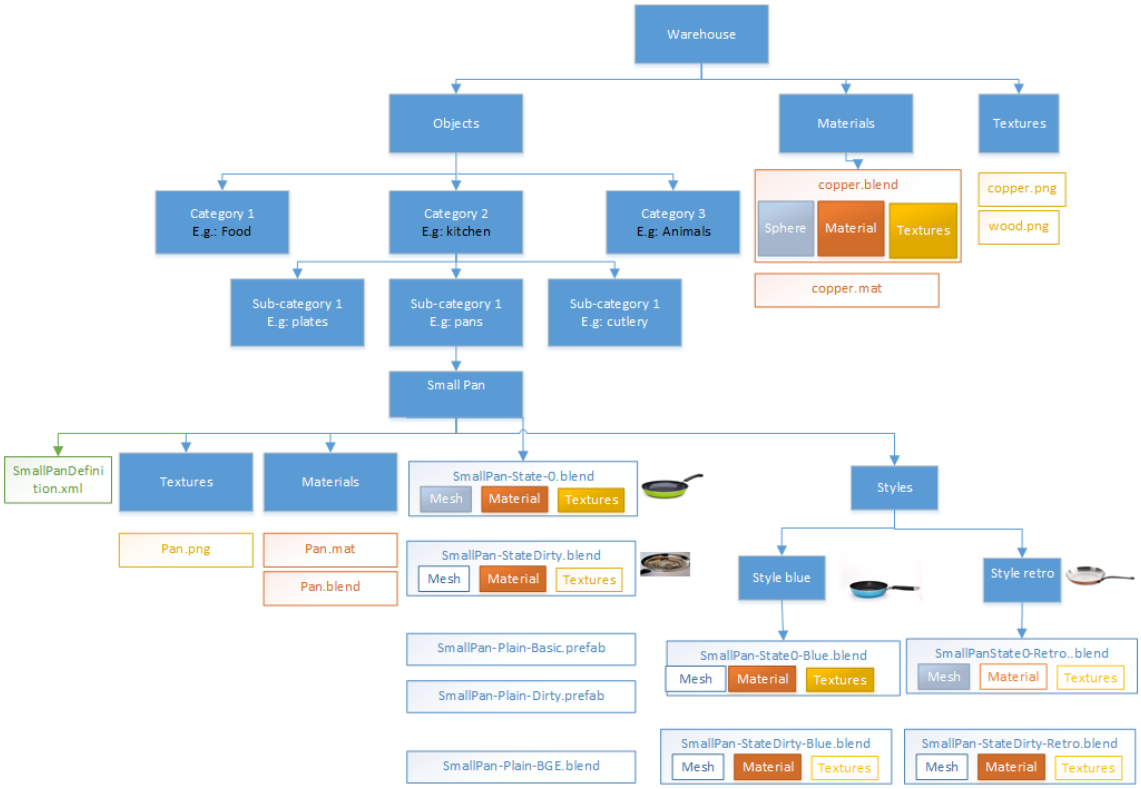

Figure 13: Structure of SLGK warehouses

Graphical models can be in any of the formats supported by Unity. The textures and materials of the different graphical models are in the local Materials and textures folders or in the common warehouse Materials and textures folders. The *styles folder contains the files needed to represent different styles of an object. For each style, there is a different folder that contains the graphical models and Unity prefabs and, eventually the Materials* and *texture* folder with the specific materials and textures of that style. Every style has all the states of the object and the corresponding graphical models.

1.1.7- Game Kit evaluation

In order to evaluate the SLGK usability and usefulness, three different use cases have been foreseen:

- Some features are missing in a currently active SeniorLudens project. A SLGK programmer is in charge of adding new objects to the scenario or modifying existing ones. The profile of the SLGK requires a good knowledge of the toolkit and of Unity. There are several subcases:
    - Adding new objects or modifying existing ones by adding visual styles, states or materials but not new actions: in this case, the world can be modified by adding already existing objects in the SL warehouses, then the designer will not need to program new features, just to set up and modify the composition of the world.
    - Adding new actions either to existing objects or new objects. In this case, the functionalities of the SLGK will need to be extended and the SLGK developer will have to program the new actions.

- A new world must be created from scratch either for a new company or for a new use case of an already existing client company.
- A new world must be created starting from an already existing Unity3D scenario, either for a new company or for a new use case of an already existing client company.

With the evaluation we aim to know the features of the SLGK designed during the first prototype needs that need to be improved and which features the developers will like to have on future versions.

The questions can be grouped in four main subjects:

1. Concepts understating and installation process: questions 1-5, 8,9, 13
2. Creation of a World: questions 12,13, 14-16,
3. Evaluation of the world creation process 17-31
4. Documentation 32-34
5. Unity 3D Expertise 35

1.1.7.1- Evaluation results

After processing the questionnaires we analysed first the questions by group, first we noticed that the developers understood how the SLGK works, and they also agree that in order to use it the developer needs a high level of knowledge of Unity3D.

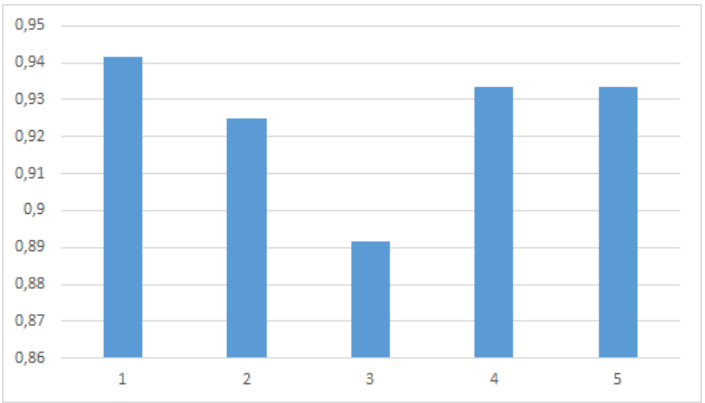

Figure 14 - Percentage of agreement

Looking for an answer for the low rating for the third group of questions, evaluation of the world creation process, we checked the features requested by the developers. Most of the

developers requested some tools to automate and work in higher lever during the creation of the worlds, they expected to have some graphical interface to design behaviours and don't have to create or modify xml files by hand or to don't create the warehouse structure manually.

To evaluate the features we categorized them in four categories:

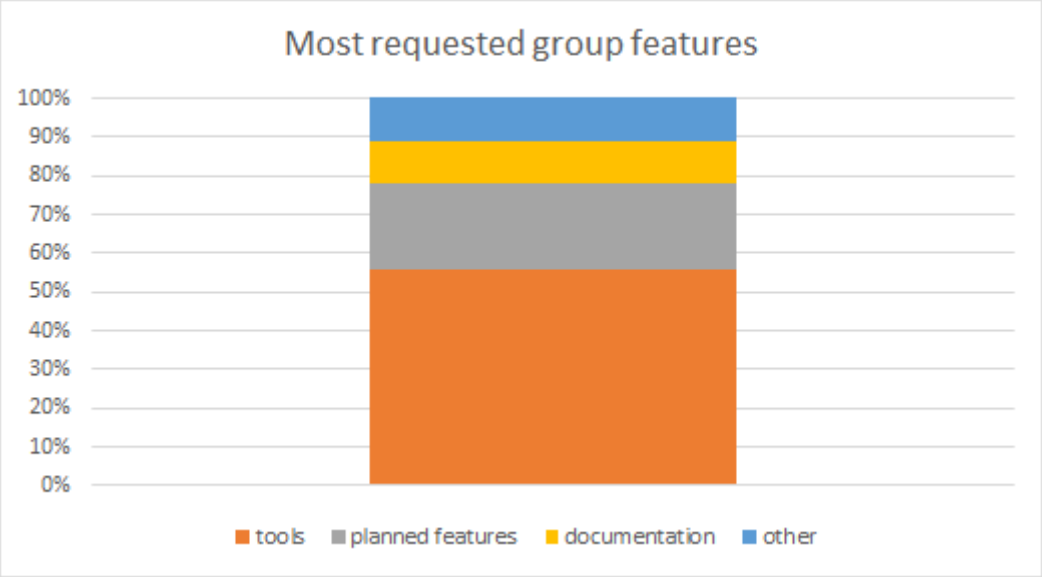

Figure 15 – Most requested group features

- Tools: this category includes the creation of tools to modify the warehouse or object definitions.
- Planned features: the user requested features that will be available in future releases of the game kit, where the most requested were: physics integration, multiple scenes.
- Documentation: provide more examples on the documentation, mostly related with the descriptors and a quick start project instead of only examples.
- Others: this group contains other features like being more verbose with the errors.

1.1.7.2- Game kit Questionnaire

The questionnaire below encompasses these different use-cases. It was used during the technical validation. It focuses at the documentation which is the main tool used by SLGK programmers.

Answer the questions below.

How would you review the following aspects of the game kit?	Totally disagree	Disagree	I don't know	Agree	Totally agree
General questions					
I was able to install the project from the SMC					
I could access to the SeniorLudens warehouse					
I understand the concept of warehouse					

Warehouses are useful					
SeniorLudens warehouse structure is practical					
I was able to create a new world from scratch					
I was able to migrate an already existing Unity model in order to create a new world					
I was able to modify an existing world					
I was able to create a new warehouse					
I was able to create a new object and add it to my new warehouse and to my world					
I was able to create a new style for an object					
I was able to create a new state for an object					
I was able to add existing actions of the SeniorLudens GameKit such as pick and drop and animate to a new object					
I was able to add a new action in the SeniorLudens GameKit					
I was able to create a message object with the SeniorLudens GameKit					
I was able to create a 2D panel with the SeniorLudens GameKit					
It was fast to create a simple SeniorLudens GameKit world with an avatar and already existing objects in it					
It was easy to create a simple SeniorLudens GameKit world with an avatar and already existing objects in it					
It was easy to add new objects to a world					
It was fast to add new objects to a world					
It was easy to modify objects of a world					
It was fast to modify objects of a world					
It was fast to add a message object					
It was easy to add a message object					
It was fast to add a 2D panel object					
It was easy to add a 2D panel object					
The currently existing actions in SeniorLudens GameKit are useful					
A lot of useful actions are missing in the current version of SeniorLudens GameKit					
The current navigation paradigm in					

SeniorLudens GameKit is suitable					
The current pick action in SeniorLudens GameKit is suitable					
The current drop action in SeniorLudens GameKit is suitable					
I could work almost on my own following the documentation					
The documentation was useful					
The creation of tickets on the issue tracker is a good mechanism to report errors and needs					
You need to be an expert in Unity3D to use the SeniorLudens GameKit					

Enumerate up to five features that the current version of SeniorLudens GameKit does not include yet and that you think that should be added in the first place in future versions

#	Description	Reason
1		
2		
3		
4		
5		

1.2- Scenario Editor

The Scenario Editor is designed as an extension of the game kit; this extension will provide new actions and objects to configure the different scenes of a world. Those actions will be available from an interface displayed over the 3D environment as it will allow the user to visualize the 3D world at the same time.

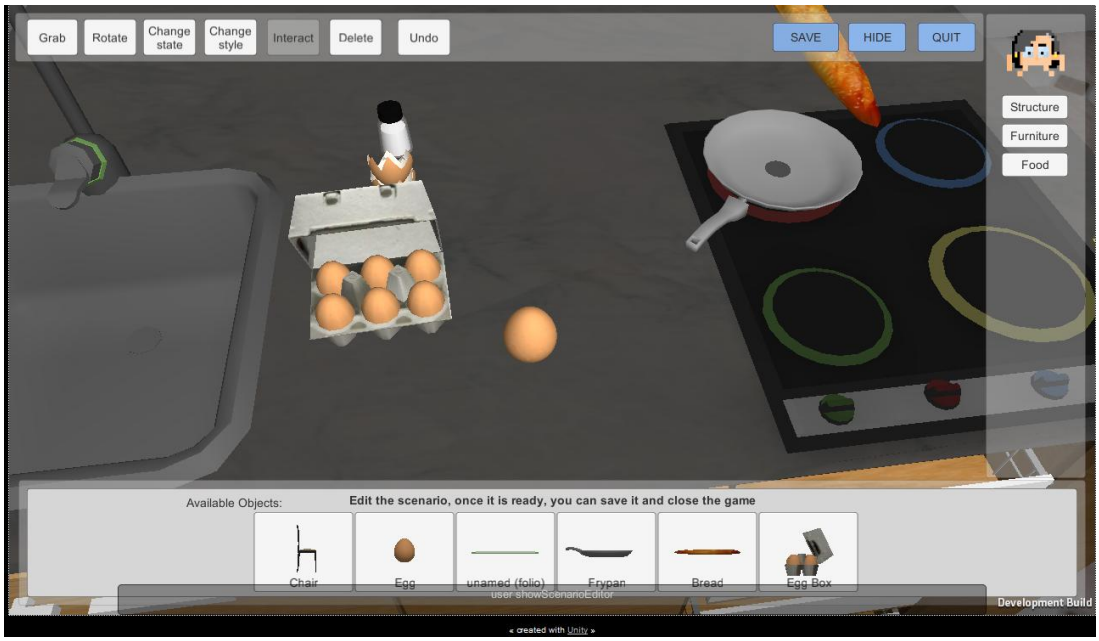

Figure 16 - Scenario Editor Interface

The interface is composed by three main areas:

- **Top bar**: the top bar contains the actions modifiers, which will modify the way the user interacts with the environment. It also provides the actions to save the scenario configuration, hide the scenario editor and to leave the game.
- **Side bar**: the side bar will allow the user to filter the existing objects of the world using categories.
- **Bottom bar**: display the objects available to place in the environment and the active message.

1.2.1- Objects

The scenario editor will need objects that are not provided by the game kit, those objects however will be mostly 2D. From the presented interface, the following objects are needed:

- **Saver**: when selected the object will save the scenario configuration and send it to the SL platform.
- **Quit**: it will close the environment, and go to the game main menu.
- **Object selector**: creates a new object and puts it on the virtual hand of the user.
- **Category selector**: when selected the bottom bar will be updated to display only the objects from a given category.
- **Action modifier**: when an action modifier is selected it will change the actions that the user can do on the 3D environment.

- **Grab**: when interacting with an object, if it can be grabbed, it will be placed on the virtual hand of the user, if not it will not change.
- **Rotate**: the selected object will be rotated by 45 degrees.
- **Change State**: the selected object will be changed to another state, for example an egg, for a broken egg.
- **Change Style**: if the selected object has more styles it will be changed for a different style, for example: a chicken egg will be replaced by a quail egg.
- **Interact**: the action provided by the disambiguation mechanism (priority table) will be used (this will be the default mode).
- **Delete**: if the user is near enough, the selected object will be removed from the scene.
- **Undo**: the previous action of the user in the scene will be discarded.

1.2.2- Actions

The scenario editor will provide new actions for the game kit:

- **Show Scenario Editor/ Hide scenario editor**
- **Save scenario Editor**: this action will export the scenario configuration in the format specified by the Scenario Configuration Descriptor and upload it to the SeniorLudens Platform.
- **Select object from Editor**: when selected a new object will appear in the scene, it will appear in the virtual hand of the user and then they will be able to place it on the desired place.
- **Undo**: this action will undo the previous action of the user, only the actions that modify the environment, like move, rotate, delete. *
- **Change to next state**: The selected object will change to another state*
- **Change to next style**: when an object is selected it will change to another of its styles.*
- **Change interaction mode**: this action will be customized for each object allowing defining which actions will be available after its activation. After its execution the way the user interacts with the environment will vary in function of which action where available. For example for “Delete” only the actions navigate, touch and delete will be available (plus the actions from the scenario editor).*

* Those actions will be implemented on the next prototype.

1.3- Task Editor

The task editor is the tool used by the Game Designer to design the reference task for the trainee and define the different roles of the characters.

The reference task is defined in terms of actions structured as sequential or parallel compositions. Sequential compositions mean that the actions must be done one after the other, and parallel compositions mean that a subset of the actions of the block must be done no matter in which order. During the game play, all user interactions are interpreted as action queries. The action queries are evaluated in comparison to the reference task to know whether they are correct or not. If they are correct, they are done. Otherwise, they can be done and evaluated as incorrect or forbidden to provide a free-of error learning process.

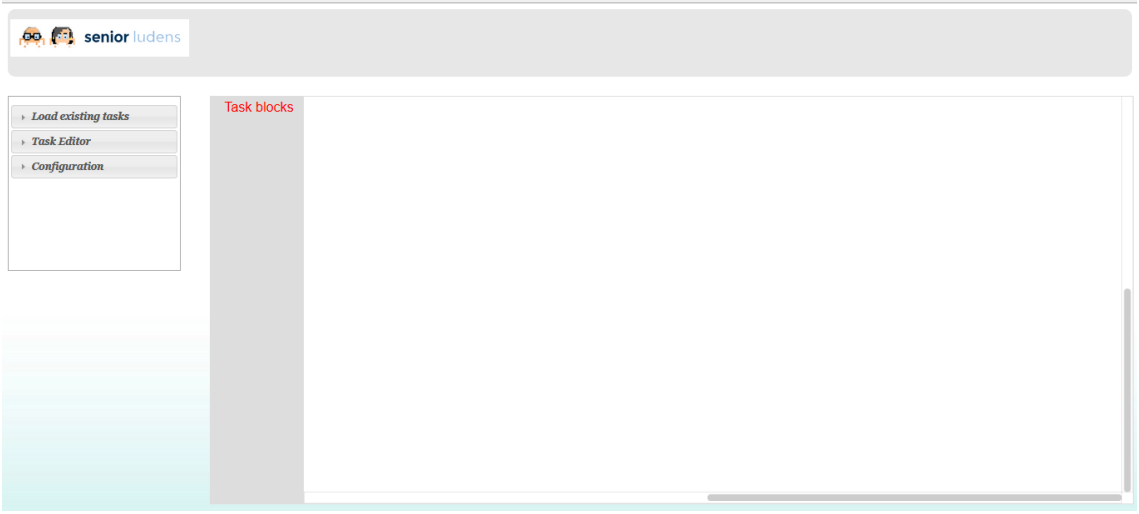

Figure 17 - Task Editor Interface

1.3.1- Implemented solution

This tool is designed to allow the users to define only the reference task, preventing them to do most of the tedious and prone to errors activities such as deploying the full state diagram of all possible user actions.

For this reasons Task Editor Tool makes use of Blockly[1], a client-side JavaScript API for Visual Programming Editor that allows users to write flows by plugging blocks together.

1.3.2- Main components

For this first release, Task editor (TE) is a web-application. In the next figure can be visualized a common java stack that are considered as model for the web development.

The java stack also includes an application Server which is in charge of publishing the TE web application developed which it covers the set of features. As application server is used Apache Tomcat

[1] https://developers.google.com/blockly/

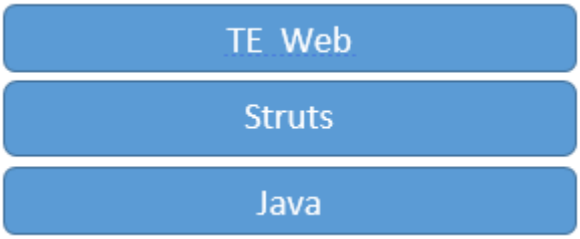

Figure 18 Task Editor Web platform - stack

1.3.3- Frontend

This element deployed over the TE Web platform provides the main frontend to the user.

It includes the principal graphical representations to the user, considered as the basic components for user interaction:

- **Manage custom blocks such as**:
  - Stage
  - Track
  - Parallel
  - Sequential
  - Action

  These blocks utilize a graphical representation provided by Blockly. They represent the basic elements used by users to create or modify the task flow.
- **Create new task:** this feature allows users to create a new task managing the set of available custom blocks included in Task editor module.
- **Modify existing task:** this feature allows users to modify an existing task managing the set of available custom blocks included in Task editor module.
- **Load existing task:** with this function the users can load and manage an existing task.
- **Create new task descriptor:** when the users have completed the task design, this feature generates an xml file descriptor able to communicate to SeniorLudens platform the interactions that should be done regarding the selected task.
- **Visualization of World descriptor:** this view utilizes the RESTful service provided SeniorLudens Storage Server. It visualizes the current World descriptor used in the task
- **Visualization of Scenario descriptor:** this view utilizes the RESTful service provided SeniorLudens Storage Server. It visualizes the current Scenario descriptor used in the task

This representation is developed over the Java stack considered as reference in the project, and consequently it is packed into a web archive that can be deployed directly over the application server.

This module does not include a data repository, because this features is provided only by Senior Ludens Storage Server.

For this reasons Task Editor manages data through RESTful calls as showing following figure:

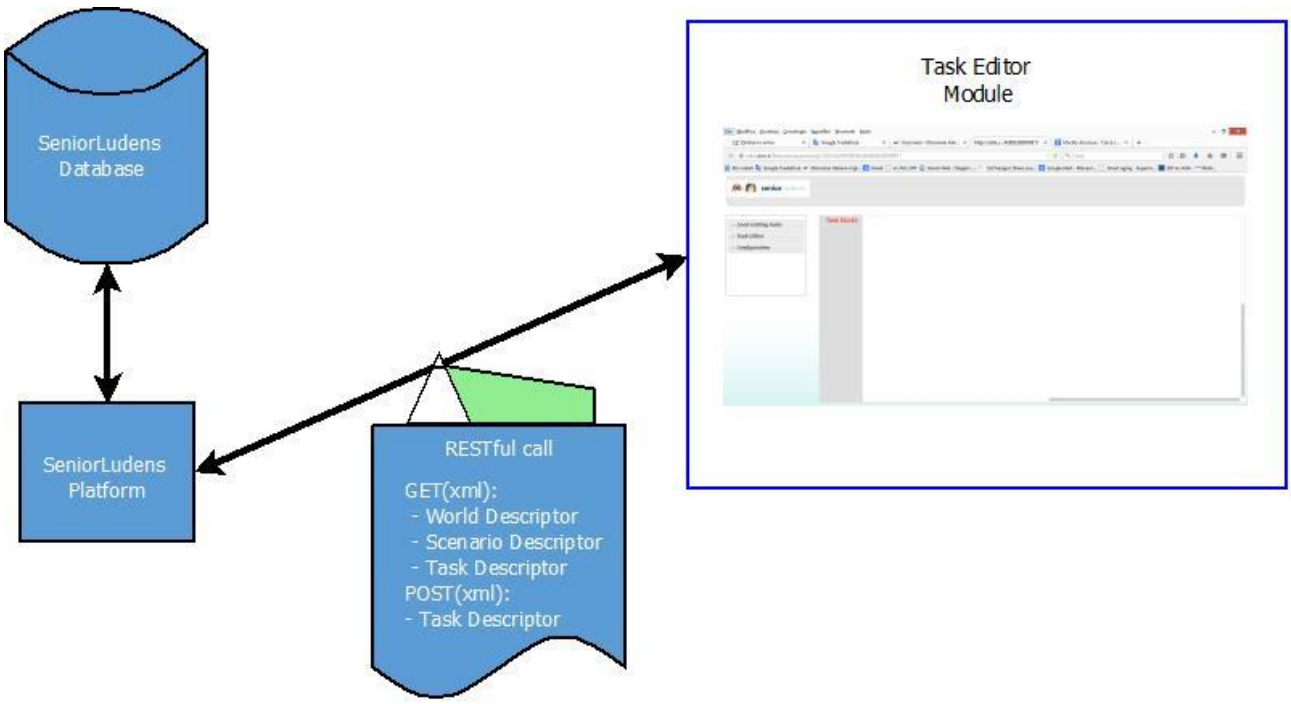

Figure 19 Task Editor data communication - flow

2- SeniorLudens Platform

SeniorLudens platform is considered as the main element where all the modules included in the project are integrated. Following this approach, it represents the main gateway with users, catching all the user actions, with a twofold objective:

- To provide access to the different systems and elements in the system
- To organize and manage the content in SeniorLudens System.

The SeniorLudens platform works closely with two other modules which shape the whole SeniorLudens System. These modules are the SeniorLudens Game Engine and SeniorLudens Storage Server.

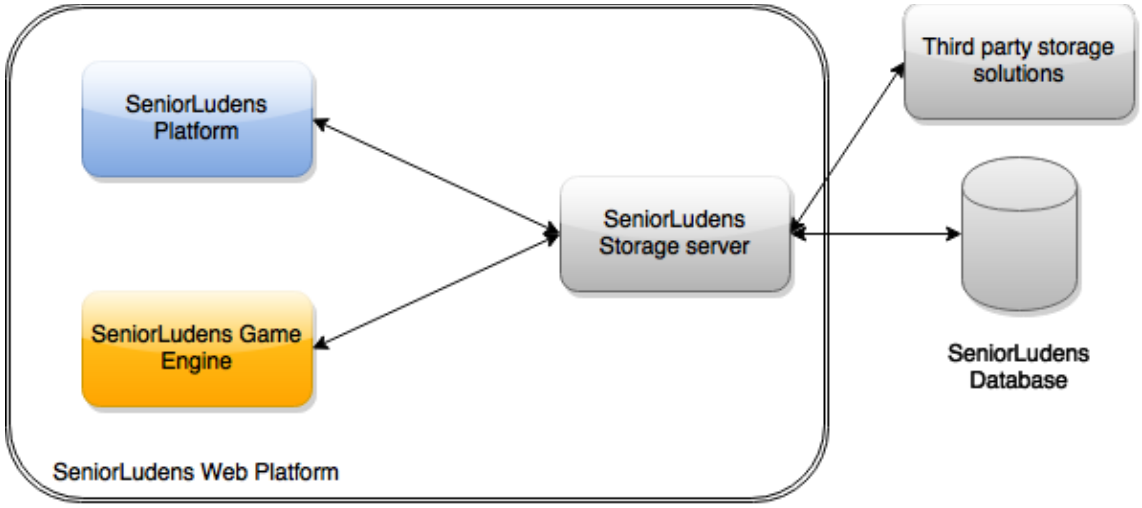

Figure 20 SeniorLudens system architecture

As can be seen in the figure above SeniorLudens Platform, SeniorLudens Game Engine and SeniorLudens Storage Server coexist in the same environment: SeniorLudens Web Platform. This environment is the web container where all the elements in the system are deployed to be available to the final users.

If we dive inside the technical architecture presented in D1.2 Technical requirements, we can distinguish how the elements are mixed between the available containers in a more detailed level. It can be remarked that not all elements in SeniorLudens Game Engine are integrated in SeniorLudens Web Platform, as they need to lay over SeniorLudens Game Kit that is the container related with the Serious Games themselves.

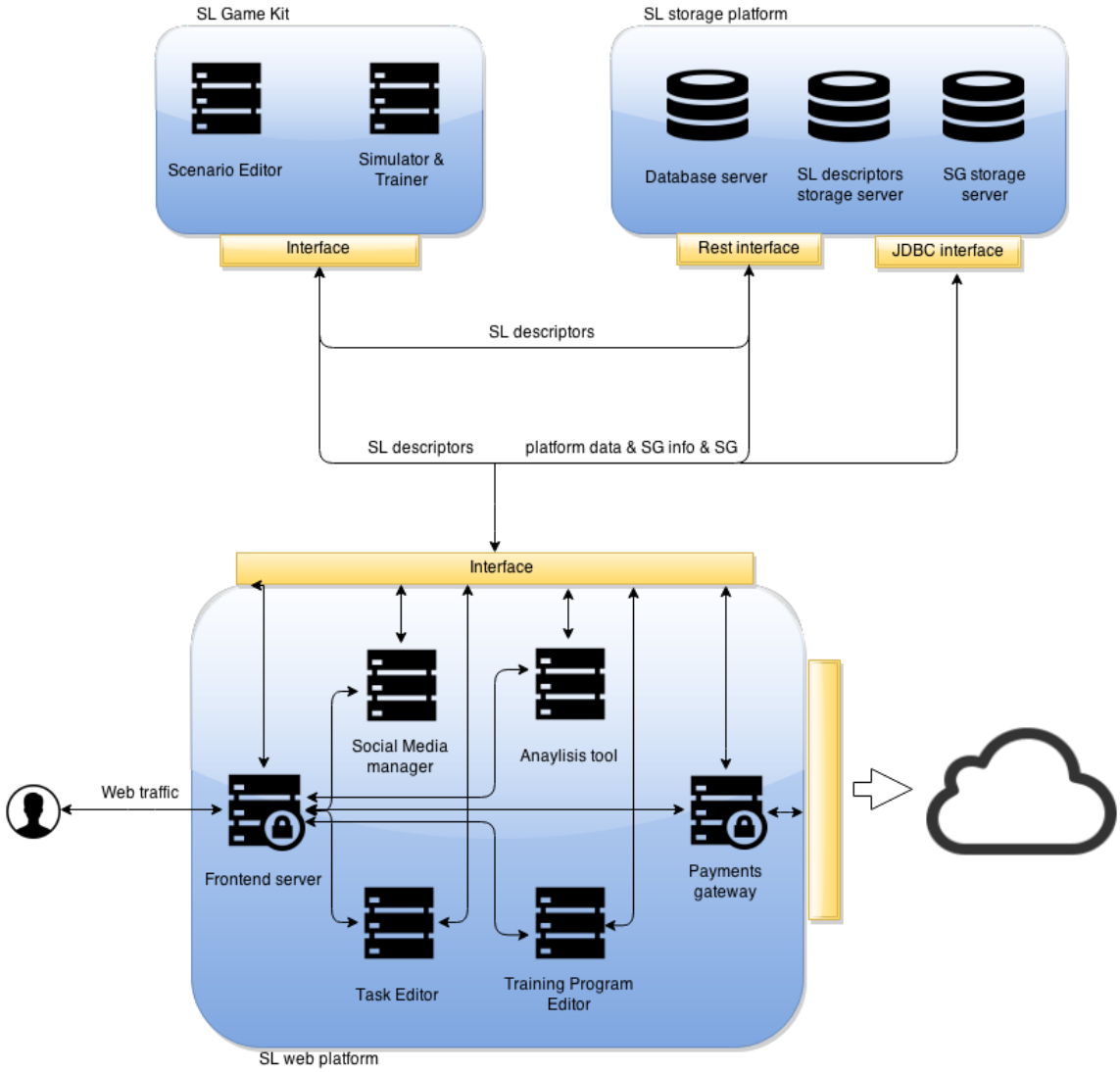

Figure 21 Technical architecture

In this document has been previously detailed the SeniorLudens Game Engine, so we will cover in this section the SeniorLudens Platform and SeniorLudens Storage Server.

2.1- Storage Server

The SeniorLudens Storage server is the element where all the data available in the platform is stored. The information can be catalogued in three different schemas depending on the context in which they belong:

- **SeniorLudens Platform Schema**: Corresponds with the Platform data.
- **SeniorLudens Descriptors Schema**: It stores the descriptors information used in SeniorLudens Game Engine.
- **SeniorLudens Results Schema**: It agglutinates the information collected from the users during the playing process. It has not been developed in the current release, so it will not be covered in the current version of the document.

2.1.1- Implementation

The storage server has a twofold implementation regarding the concept of the system, as it includes storage elements and a server to provide the connectivity with the clients to interchange the store information.

The storing subelement is conceived using a PostgreSQL database as the main storage base. The selection of this database was based on the licensing terms which apply to it and because of the direct integration with PaaS solution where the final solution is deployed.

The server side of the Storage Server is based on the java default stack defined in the project. Being the server intended only to provide mechanisms to access data, as it is not needed to build web interfaces. This reason motivates the utilization of Java2EE in the main core and Spring to provide the mechanisms manage the server and the connections with the clients. It is also used Hibernate to manage the connections with the databases.

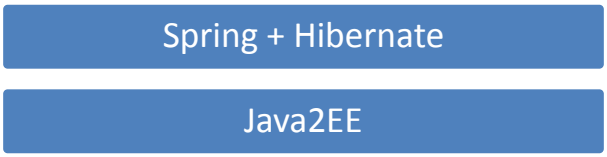

Figure 22 Storage Server stack

There are defined two interconnection procedures to attend the users' calls depending on the specific internal client which is performing the call.

- **Rest API**: This is the default connection method. By using it, there are provided a transparent api methods that can be queried to consume the stored data. It is used to retrieve the data stored in SeniorLudens descriptor schema and SeniorLudens Results schema.
- **JDBC connection**: This link is used by the systems which want to connect directly with the data stores. It requires guaranteeing the integrity of the queries executed in order to maintain the data consistency in the store. It is used by SeniorLudens Platform schema because it is needed to satisfy the time constraints on the responses to the user interactions.

2.1.1.1- SeniorLudens Platform schema

This schema is used to store all the information managed by the SeniorLudens Platform, so it includes the information about the users, organizations, games, game versions, etc.

This database is accessed through a JDBC connector in order to provide all the information directly from the storage to the SeniorLudens Platform.

The next figure shows the class diagram involved in the SeniorLudens Platform to deal with the data stored in the schema. This model correlates uniquely with the entity-relationship design of the schema.

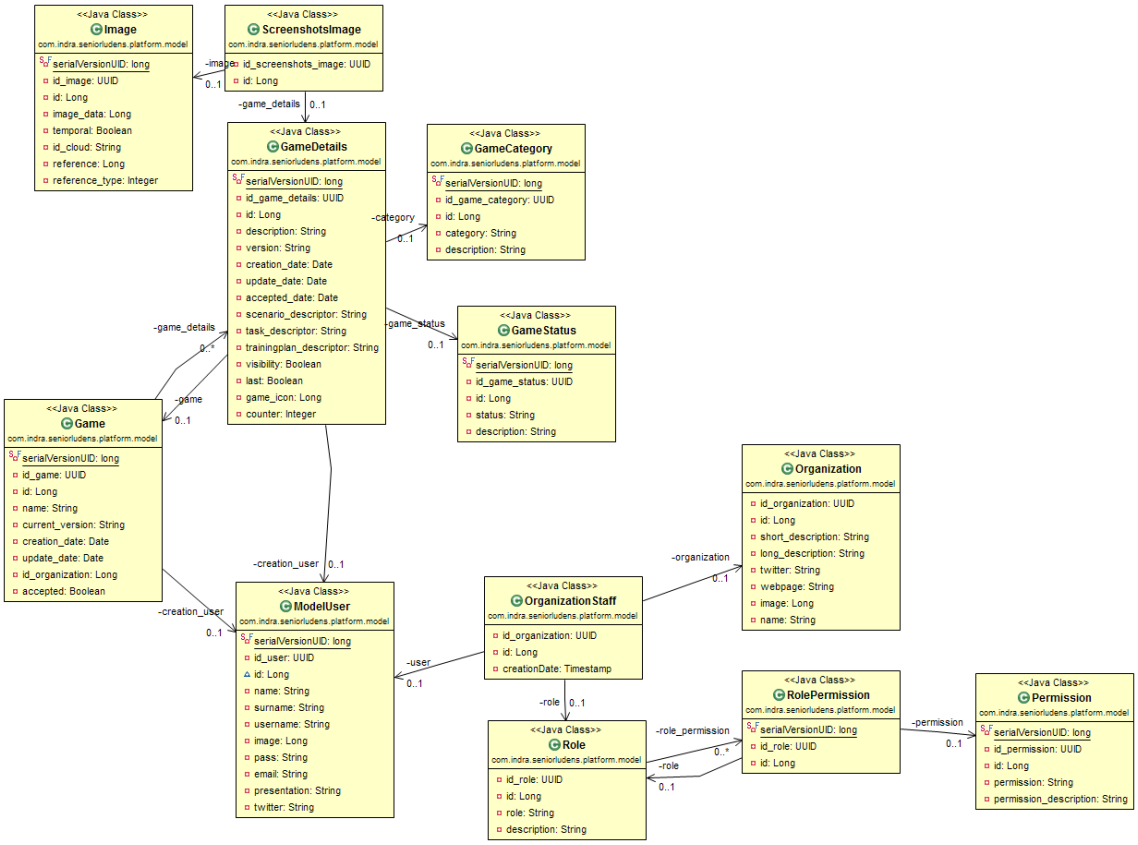

Figure 23 Data model class diagram

The elements involved in the model are the following:

- **Game:** This table stores the information about the game in a general level. Only information not intended to be changed among the different game versions is included in this table.
- **Game Details:** The game details corresponds with the different game versions which a Serious Game may have. The last version should be the most recent in the game Catalogue.
- **Game Category:** This table stores the game categories available in the system. All games in the catalog must be fitted in one of these categories.
- **Game Status:** The table provides the different possible status in which a game can be. It contemplates options to publish, unpublish, review, etc.
- **ModelUser:** This table provides the model for the users in the system. It includes all the personal data needed to identify uniquely the users in the system. The password is not stored in plain text, as it is required to be coded to guarantee a minimum level of security under unauthorized access to database.
- **Organization Staff:** This table organizes the different roles and users which comprises the organizations staff.
- **Organization:** These are the organizations included in the system. This table includes all the information required to identify uniquely an organization in the system.
- **Role:** This table stores the different available roles that a user may have in the system. The link between roles and users is done in the table Organization Staff.
- **Role Permission:** This table is in charge of tying roles and permissions together. By defining these connections, the platform knows which roles have access to individual actions.

- **Permission:** This table defines the individual actions referred to the platform. It is considered as the minimum definition that can be used to define the operations that can be performed in the SeniorLudens Platform.

2.1.1.2- SeniorLudens descriptors schema

This database is the place where all data related with the descriptor files used in SeniorLudens Game Engine are stored. These descriptors define uniquely a Serious Game in the system and they are needed to execute the game. Because of this, it is highly important to provide secured mechanisms to guarantee the data integrity as well as standard mechanisms to access the data.

In order to compel the integrity on the data, this database is only accessible through REST APIs, which guarantees the communication based on a defined handshake and error returning codes. This mechanism ensures the validity on the data and the knowledge about the state of the communication in both sides involved in it.

The data integrity is accomplished by checking all the information stored together with the new descriptor files coming from the clients, with the specific XML schemas which models the descriptors (each descriptor has its own schema which models it). By passing successfully through the schema check, the descriptor is able to be stored or sent to the client. If the descriptor does not pass the check, then it is returned a negotiated error code, to let the client know about the existing problem in the server. This procedure ensures the data consistency, avoiding corruption on the descriptors and hence over the Serious Games.

The database schema responds to the class model shown in the figure below:

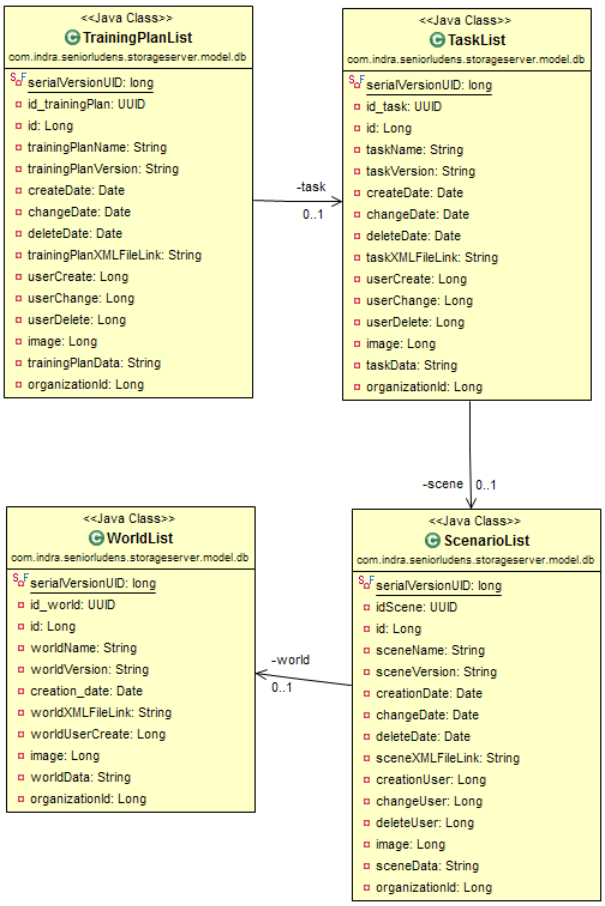

Figure 24 Data model descriptors class diagram

The class diagram shown in this figure corresponds with the model defined in the SeniorLudens Storage Server, which correlates directly with the entity-relationship design used in the schema.

The main elements in the model are the following:

- **WorldList:** This table stores the world descriptors. It also connects the world descriptor files with the owner organization in order to provide traceability on the worlds designed by the owner entities. This table stores audit data to track the changes on the data as well as the users involved in the operation.
- **ScenarioList:** This table stores the scenario descriptors. It has direct connection with the WorldList table as it builds over world hierarchically. It includes the organization ownership information and the audit information as well.
- **TaskList:** The table stores the task descriptors. It has direct connection with the ScenarioList because the descriptor has complete meaning only under the selection of a scenario descriptor. In the same way as the other tables, it includes the information about the organizations and data for auditing purposes.
- **TrainingPlanList:** This table stores the information about the training plan descriptors. Following the same approach of the other tables, it needs the Task Descriptor to have complete meaning. It provides information about the organization and for auditing purposes as well.

It is expected to update this model in the next releases to replicate the descriptor files in third party storage solutions, so it is planned to update the model in the next development iterations.

As was commented before, the communication method defined for this database schema is the REST API. It is defined a set of methods to retrieve the information stored based on the different context on which they apply. The conjunction of these methods and the XML schemas used for validate the data consistency on the descriptors, shapes the SeniorLudens Storage Server for serving the descriptors data.

Regarding the world descriptors have been defined the following methods:

Endpoint	Method	Parameters	Description
https://seniorludens-pre.herokuapp.com/rest/api/storage/world	GET	Long: id (the world id)	It retrieves from storage server the requested world descriptor.
https://seniorludens-pre.herokuapp.com/rest/api/storage/world	POST	world: String (The world xml data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name)	It stores a new world descriptor into storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/world/update	POST	world: String (The world xml data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) id: Long (the descriptor id)	It updates the world descriptor in storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/world/organization	GET	organization_id: Long (The owner organization)	It gets the descriptors owned by the organization id.

Table 1 REST methods for world descriptors

The scenario descriptors stored in the server are accessed by the following methods:

Endpoint	Method	Parameters	Description
https://seniorludens-pre.herokuapp.com/rest/api/storage/scenario	GET	Long: id (the scenario id)	It retrieves from storage server the requested scenario descriptor.
https://seniorludens-pre.herokuapp.com/rest/api/storage/sc	POST	scene: String (The scene xml data)	It stores a new scenario

enario		organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) world_id: Long (The world id from where it hangs)	descriptor into storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/scenario/update	POST	scene: String (The scenario xml data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) id: Long (the descriptor id) world_id: Long (The world id from where it hangs)	It updates the scenario descriptor in storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/scenario/organization	GET	organization_id: Long (The owner organization)	It gets the descriptors owned by the organization id.
https://seniorludens-pre.herokuapp.com/rest/api/storage/scenario/organization/{world_id}/	GET	world_id: Long (PathParam)(the parent world id)	It gets the set of scenario descriptors hanging from world descriptor id.

Table 2 REST methods for scenario descriptors

The task descriptors are available using the following methods:

Endpoint	Method	Parameters	Description
https://seniorludens-pre.herokuapp.com/rest/api/storage/task	GET	Long: id (the task id)	It retrieves from storage server the requested task descriptor.
https://seniorludens-pre.herokuapp.com/rest/api/storage/task	POST	task: String (The task xml data) organization_id: Long (The owner organization) user_id: Long (responsible user)	It stores a new task descriptor into storage server.

		name: String (descriptor name) scene_id: Long (The scene id from where it hangs)	
https://seniorludens-pre.herokuapp.com/rest/api/storage/task/update	POST	task: String (The task xml data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) id: Long (the descriptor id) scene_id: Long (The scene id from where it hangs)	It updates the task descriptor in storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/task/organization	GET	organization_id: Long (The owner organization)	It gets the descriptors owned by the organization id.
https://seniorludens-pre.herokuapp.com/rest/api/storage/task/organization/{scene_id}/	GET	scene_id: Long (PathParam)(the parent scenario id)	It gets the set of task descriptors hanging from scene descriptor id.

Table 3 REST methods for task descriptors

The training plan descriptors can be queried using the following methods.

Endpoint	Method	Parameters	Description
https://seniorludens-pre.herokuapp.com/rest/api/storage/trainingplan	GET	Long: id (the training plan id)	It retrieves from storage server the requested task descriptor.
https://seniorludens-pre.herokuapp.com/rest/api/storage/trainingplan	POST	training: String (The training plan xml data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) task_id: Long (The task id from where it hangs)	It stores a new training plan descriptor into storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/tr	POST	training: String (The training plan xml	It updates the training plan

ainingplan/update		data) organization_id: Long (The owner organization) user_id: Long (responsible user) name: String (descriptor name) id: Long (the descriptor id) task_id: Long (The task id from where it hangs)	descriptor in storage server.
https://seniorludens-pre.herokuapp.com/rest/api/storage/trainingplan/organization	GET	organization_id: Long (The owner organization)	It gets the descriptors owned by the organization id.
https://seniorludens-pre.herokuapp.com/rest/api/storage/trainingplan/organization/{task_id}/	GET	task_id: Long (PathParam)(the parent task id)	It gets the set of training plan descriptors hanging from task descriptor id.

Table 4 REST methods for training plan descriptors

It is important to remark that all the methods listed above are constructed by using the SeniorLudens staging environment URL as root (http://seniorludens-pre.herokuapp.com). The server can be deployed to any server, so the only change in the endpoints methods will consist in change the root of the shown methods.

All the methods listed on the previous tables will be updated in the next development phases in order to adapt them to the necessities of the next deployments of the different elements on SeniorLudens Game Kit.

2.2- SeniorLudens Platform

The SeniorLudens Platform comprises the web interface used in the project to interact with the users. It also agglutinates the access to all the elements in SeniorLudens System, being considered as the common entrance point between the users and the different tools integrated in the system.

It is deployed in a web container that is used by all the web servers and web applications. It has been done in this way due to the PaaS deployment used in the environments. However all the web apps can be splitted to be deployed in different containers and host servers in order to scale the system to the user load or simply to match the user experience based on QoS agreements.

2.2.1- Implementation

The platform has been built using the reference stack defined in the project. The bottom level is J2EE in which all the rest of the stack levels are piled up. The reference layer is constructed over Spring and Maven. Using this architecture, is pursued to offer a scalable project

management, and adaptable to different environments what is valuable when the development is deployed in development, staging or production systems.

Going into detail in the Spring module, it uses Spring Boot as reference to provide the deployment contexts on the web container. To respond to the views it is used Spring MVC which gives to the system the scalability on view adaptation and rest method development. The security in the system is provided through Spring Security, which facilitates the security methods during the user's sessions and login views.

The project is deployed over Tomcat 8.0 web container, in which it is executed the platform web application. The views are composed by Apache Tiles, taking advantage of the integration of different JSP pages to shape the final view. It is really handy because it maximizes the reuse of components inside the project, reducing consequently the redundancy in project sources. The final presentation layer is built over HTML, JavaScript, and CSS3. This layer is constructed using Bootstrap solution in order to ease the responsive web integration. It makes the final result more adequate and comparable with the current solutions deployed in the web.

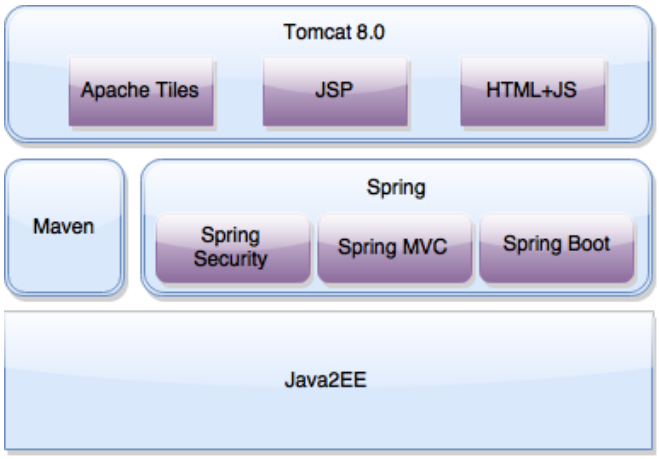

Figure 25 SeniorLudens Platform stack

Using this stack as development reference, the platform aims to get a minimalistic approach in its visualization, stressing by the use of vivid colours to guide the user to the desired actions. Besides the views have been organized using the same schema: main actions are located in the main part of the screen; other actions and tools available in the system on the side menu; and additional functions to change the status of the platform on the top menu.

It has been considered a responsive design in all the views defined in the platform. It ensures the correct visualization in all devices: computers (considered as the main device which is going to access the platform), tablets and smartphones. By using the responsive design, all the elements in the view are laid out to compose a new arranged view in which the visibility and accessibility are maximized depending on the new screen resolution.

Figure 26 SeniorLudens platform

The development has followed a Model-View-Controller approach, in which the view is provided by the JSP and Apache Tiles, the controllers that are in charge of responding the user's requests, done over the views, and the Model that corresponds in the case of the Platform to the model defined in Figure 23. This model correlates directly with the model used in the definition of the database schema.

From this point we are going to dive in detail on the specific construction of the system through the different layers and classes. All the design is based on interfaces and inversion of control mechanisms (IoC), with the objective of create a scalable and transparent between layers infrastructure.

The controller's class diagram can be visualized in the next figure. The controllers are supported by the service's classes in order to organize and distribute the access to the dao classes. Using this approach: views talk with controllers; controllers talk with views and services; services provide the business logic and access to the dao classes; and dao classes talk with database.

It can be distinguished the following main elements:

- **Register Controller:** This controller is in charge of the register calls attached to the register view. It assists as well during the new user creation procedure. It is supported by the User Service, to update the status of the new user in the system.
- **Login Controller:** This controller controls the login features and views in the system. It is strictly connected with the system security to guarantee the data interchange in the user session. It has direct connection with the Organization Service to retrieve the user organizations, Game Service to obtain the games in each one of the organizations in where the user has an admin role access and User Service from where the controller takes the information about the user model object.
- **User Controller:** This controller manages all the connections to update user profile, as well as to update the user role in the organizations. To do so, it has direct connections with UserService, to access the users' data; Organization Service, to get the information about the organizations; and RoleService, to retrieve the information about the user roles in the system.

- **Game Controller:** The game controller is in charge of managing all the connections with the views related with game catalogue management. It has direct access with Game Service to query about the game model objects in the system; Organization Service to get the information about the organizations which own the games in the system; and User Service, to obtain the information about the users who were involved in the game lifecycle.
- **Organization Profile Controller:** This controller manages all the views related with the individual organization profile. It connects with the Organization Service to obtain the information about the organizations registered in the system.

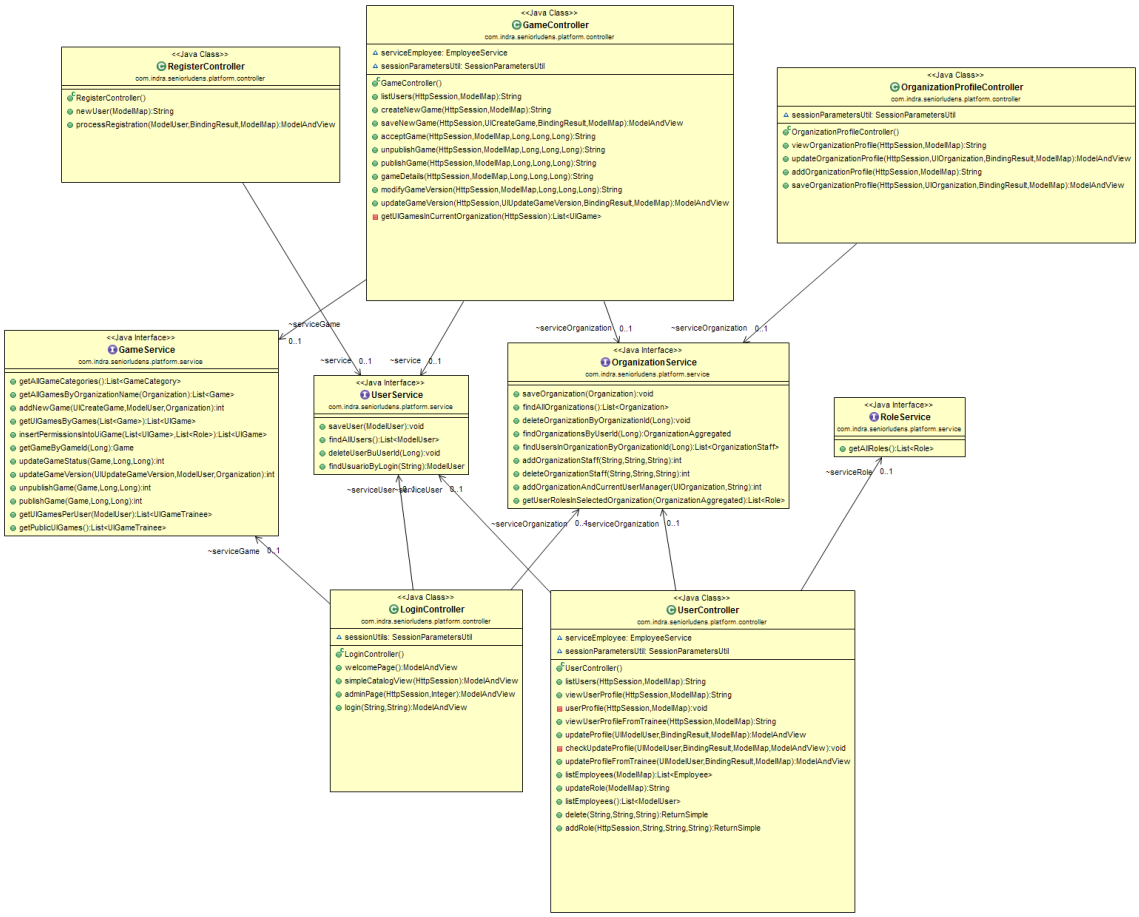

Figure 27 Class Diagram – Controllers & Services

As was aforementioned, the bottom layer in the project class structure corresponds with the DAO classes which are in control of the database access. Any consultation/update/delete operation on database is done only by using the DAO objects. This uniqueness in the access mechanism guarantees the independence of the data and the stability in data accesses, as all sources of consultations use the same data access methods.

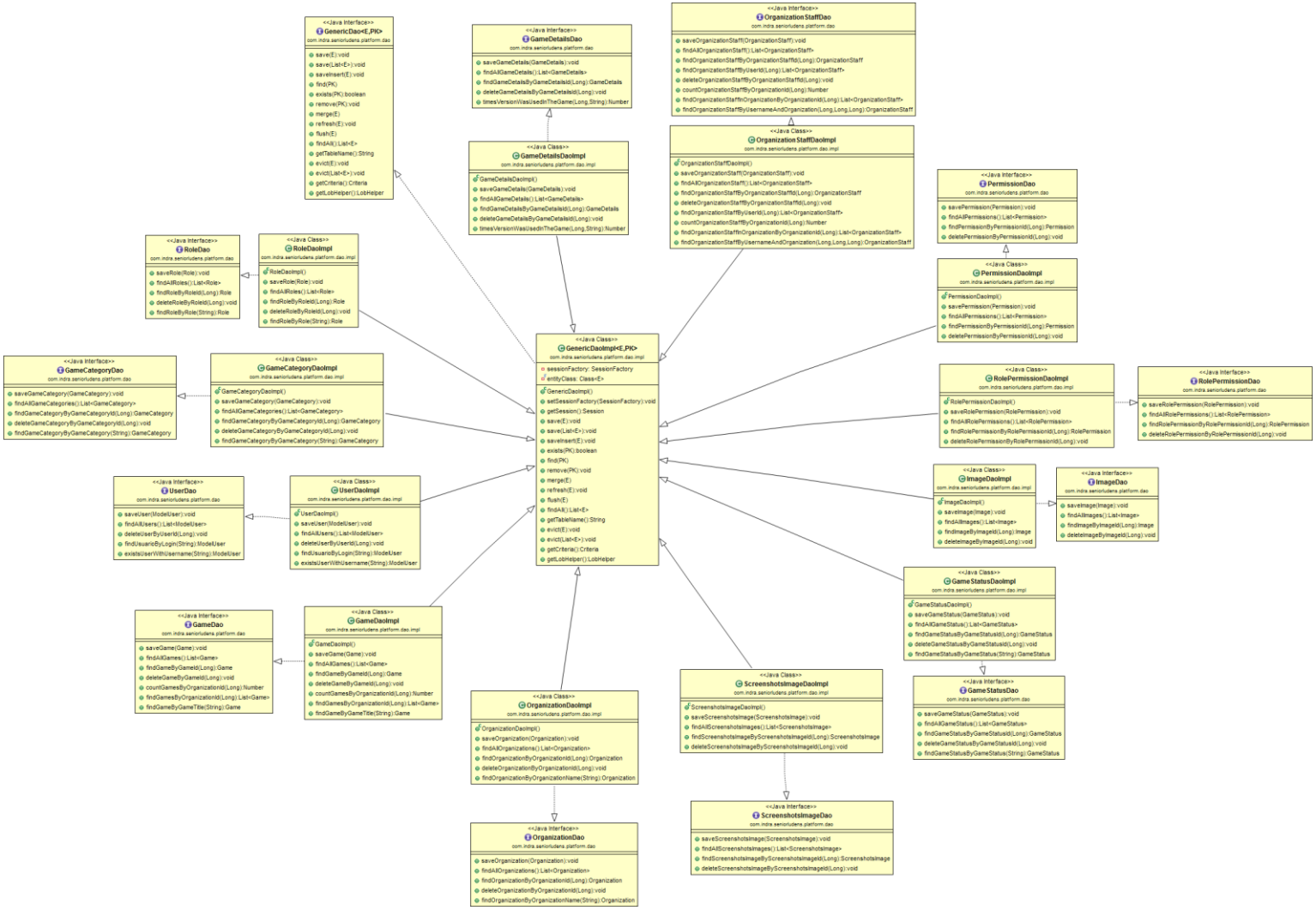

Figure 28 Dao Class Diagram

As can be seen in the class diagram, all DAO classes extend from GenericDaoImpl, which encapsulates and provides the basic methods for data select/update/delete (CRUD methods). By extending this class, all DAO classes use the same connectors with the data source. By combining these methods, all DAO classes can create extended methods regarded to each entity.

The class diagram is shown in the figure above. In it can be observed that there is a direct connection between data model and DAO classes, by which a model class is mapped by a DAO class which encapsulates all the connections. Can be highlighted the following main classes:

- **Game DAO:** It is connected with Game model class. It is in control of the game information in database, so it provides the methods to interact with the Game model.
- **GameDetails DAO:** It is connected with Game Details model class. It provides the methods to query about the different version of the games.
- **GameCategory DAO:** It connects with Game Category model class. It provides access to the game categories in database. This content is almost static because the categories are unalterable during the platform execution.
- **GameStatus DAO:** It is connected with the Game Status model class. It gets access to the different states in which a game can be in the platform. The content of this DB is static during the platform execution.
- **User DAO:** This class connects with the User Model class. Besides it provides access to the basic methods to operate with the user fields.
- **Organization DAO:** It connects with Organization model class, offering access to the information contained in the organizations profile.
- **OrganizationStaff DAO:** It connects with the Organization Staff model which is in charge of structuring the user roles inside the organizations. This class presents the methods to access this information.
- **RolePermission DAO:** It connects with RolePermission model, which is oriented to join all the different basic actions over the platform into Roles. These roles can be understood as sets of actions.
- **Permission DAO:** It is the direct access class to Permission Model. These data is almost static in the project, as it defines all the basic actions that can be performed over the platform.

2.3- Evaluation

The evaluation of the platform is carried out by matching the current development with the technical requirements defined in the project. The functional requirements are going to be tested and described in D4.2 Pilots evaluation results.

The platform and storage server have been deployed in Heroku[2] PaaS solution following the definition of the technical requirements done in the deliverable D1.2. The decision of this provider has been taken based on the conditions, pricing plans and capabilities offered by the different competitors in the market. After this evaluation process, it was considered Heroku as the best option to deploy the different releases of SeniorLudens project.

This decision is also compatible with the necessity of creating different environments depending on the stage in which the project and its releases are: development stage, staging stage and production stage. Depending on the different stage of the Continuous Delivery in which the current versions are, there will be selected the suitable environment. For the moment of the redaction of the present document, the two main environments for the platform in the development phase have been created: development and staging. The current release is in fact

[2] www.heroku.com

deployed in the staging environment, and it is being updated continuously with new updates and releases taking the benefits from continuous delivery approach.

The present status of the platform and storage server includes all the elements commented in the document. It is under development after the first iteration in the development and the next iteration will cover the extension of the existing features and will go ahead in the integration of the different systems (especially SL Game Engine) and the tools in the platform.

Figures and tables

Acronyms

Acronym	Explanation
SL	SeniorLudens
SLGK	SeniorLudens Game Kit
SDK	Software Development Kit
LMB	Left Mouse Button
RMB	Right Mouse Button
QoS	Quality of Service
DAO	Data Access Object
IoC	Inversion of Control
CRUD	Create, Read, Update, Delete
DB	Database