AMBIENT ASSISTED LIVING JOINT PROGRAME

AAL-2013-6-039

SeniorLudens

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

D2.5 Serious Games development engine (1)

Workpackage	WP2 – Serious games development engine 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)	Marije Blok – KBO 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 aims for detailing the installation of SeniorLudens Game Engine. It can be also considered as a user manual attached to the software pilot developed.

The Game Engine is the system in charge of automating the Serious Games creation process. Based on the different element which comprises the SL Game Engine, this deliverable is composed of three parts: in the first part it describes the user manual of the Game Kit (SLGK), in the second part the Scenario Editor and in the third part the Task Editor.

All the information described in the document is available in html to support the organizations and game designers during the game creation process.

1- Senior Ludens Game Kit Documentation

SeniorLudens is a Serious Game Development Platform for older workforce training and inter-generational knowledge transfer, funded by the EU program AAL-2013-6-039. The platform is formed by multiple components and one of them is the GameKit, this documentation is about it.

To know more about the platform and the role of the Gamekit in the platform, please read SL_Gamekit.

This part introduces the SeniorLudens GameKit and then focuses on step- by-step instructions for the creation of games.

1.1- Installation

SeniorLudens GameKit currently only supports one game engine: Unity 3D. This section explains how to install the game kit and its dependencies in a Unity project.

Note: Please take note that, in the current stage of SeniorLudens development, the SeniorLudens GameKit and warehouses are installed in intermediary servers. This will be modified in the future since everything will be installed in the final SL server. Therefore, these instructions will need to be updated when the SeniorLudens GameKit will be uploaded to the SeniorLudens platform.

1.1.1- Install the Gamekit for Unity

1.1.1.1- Requirements

To use the game kit, you must install the following programs:

- Unity 3d 4.64f
- Git
- Python >= 3.3 1
- Blender 2

Why Unity 4.6.4f1?

The reference version of Unity 3D for developing games with the SeniorLudens GameKit is 4.6.4f1 (99f88340878d), however any minor version from the 4.6 branch should work and we recommend to use the latest version from this branch before opening to the public a new game.

- **Download links** *Windows Mac*
- **Release notes** *4.6.4*
- **API** *http://docs.unity3d.com/462/Documentation/ScriptReference/index.html*
- **API History** *http://docs.unity3d.com/ScriptReference/40_history.html#4-6-0*

The development of the game kit started with version 4.5.0 (27 May 2014), since the announcement of the new UI on version 4.6 we were expecting it. We tried it just after the release, on April, and we found some issues. We decided to wait few minor versions, finally we migrated from 4.5.5 to 4.6.3, few weeks later unity announced the public release of a new major version 5, at this point we decided to not upgrade to 5 and stay with 4.6 until autumn, we did a minor upgrade to 4.6.4 and due to the fast release cycle of the following minor versions we decided to not.

Note: We plan to migrate to Unity 5 during September 2015, or at least upgrade to the newest minor version of the 4.6 branch.

1.1.1.2- Obtain the Game kit

Warning: Once the game kit becomes more mature, we will provide it as an asset importable to Unity. By the moment the unique way is to obtain it directly from its repository.

You can obtain the game kit for Unity cloning the repository SeniorLudens UnityGameKit:git@movibio.lsi.upc.edu:seniorludens/unity.git inside your Assets folder, with the following commands:

```
# Clone the gamekit
git clone git@movibio.lsi.upc.edu:seniorludens/unity.git SeniorLudensGameKit
# Init gamekit submodules
cd SeniorLudensGameKit
git submodule update --init
```

Note: If your project uses git as source management control, you will prefer to add the gamekit as a submodule 3 instead of cloning the repository:

```
git submodule add git@movibio.lsi.upc.edu:seniorludens/unity.git Assets/SeniorLuden
# Init gamekit submodules
cd SeniorLudens
git submodule update --init
```

1.1.1.3- Obtain the Game kit SDK

The SeniorLudens Gamekit SDK extends the Unity Editor to improve the development experience with the SeniorLudens GameKit, its features are explained in the section gamekitunity3d.

Note: The following code expects to have Python installed, we recommend using Python 3.4 (amd64).

To install it you have to clone the SeniorLudens UnityGamekitSDK: git@movibio.lsi.upc.edu:seniorludens/unitysdk.git to your Assets/Editor folder, if you are inside your assets folder you can do it with:

```
# Clone the gamekit
git clone git@movibio.lsi.upc.edu:seniorludens/unitysdk.git Editor/SeniorLudensGame
pip install -r Editor/SeniorLudensGameKit/requirements.txt
```

Or if you prefer to use a submodule:

```
# Clone the gamekit
git submodule add git@movibio.lsi.upc.edu:seniorludens/unitysdk.git Assets/Editor/S pip
install -r Assets/Editor/SeniorLudensGameKit/requirements.txt
```

Finally you have to install manually lxml 4, if you are using MS Windows, we recommend you to use an unofficial build of lxml:

```
pip install "http://www.lfd.uci.edu/~gohlke/pythonlibs/3i673h27/lxml-3.4.4-cp34-non
```

1.2- Create a new world

1.2.1- Before you start

Every time a new virtual environment is needed, a new SeniorLudens world (see SL_worlds) must be built. These instructions explain how to do it. If you already have a world and you want to add it some objects or actions, go to “Modify a world” section. If you have a Unity3D scenario that you want to use in SeniorLudens, follow to “Migrate a Unity Scenario” section. In any case, before you start, read Section Who is able to modify a world and why? subsections to make sure that you are aware of the roles and permissions in the SeniorLudens platform.

Creating a world means not only creating a 3D scenario, but also organizing it in the very specific way that SeniorLudens GameKit needs in order provide the required functionalities. In particular, you will need to create a new Unity3D project, do some settings, create a warehouse (see warehouses), scenes 1 and add the basic structural objects. After that, the world core will be created and you will be able to improve it adding objects and behaviours following the instructions of Modify a world.

Please, follow the instructions sequentially, otherwise you may end in a deadlock.

1.2.2- Instructions

1.2.2.1- Basic setting for the creation of a new world

In this document, you will learn how to pull the needed files from the git repository and how to create a new Unity3D project for your game.

1. Open http://movibio.lsi.upc.edu/gitlab/groups/seniorludens
2. In the right menu, select New Project (green button).
3. Follow the step-to-step instructions: specify the path of the project and a short description. The namespace is seniorludens (default) and the scope is private. Confirm.
4. Open a new git window or console and follow the instructions step-by-step: general setting the first time and create a repository.
5. Create a directory called src the project directory, this will be the folder the Unity Project.
6. Open Unity -> File -> New Project and indicate the src directory.

7. Automatically, various directory will be created inside src, one of them called Assets will contain all the resources of the project.
8. Download the submodules: SeniorLudens/warehouseSL and SeniorLudens/Unity inside the directory Assets. To do so: in a terminal, from the project's root directory, write: git submodule add <path of the submodule inside gitLab> Assets/file_name. Download core and *unity in a common directory that you will call SeniorLudens. specifically, do:

   git submodule add git@movibio.lsi.upc.edu:seniorludens/warehouseSL.git src/Assets/w git submodule add git@movibio.lsi.upc.edu:seniorludens/unity.git src/Assets/SeniorL git submodule add git@movibio.lsi.upc.edu:seniorludens/unitysdk.git src/Assets/Edit

9. In the directory src/Assets/SeniorLudens, do:

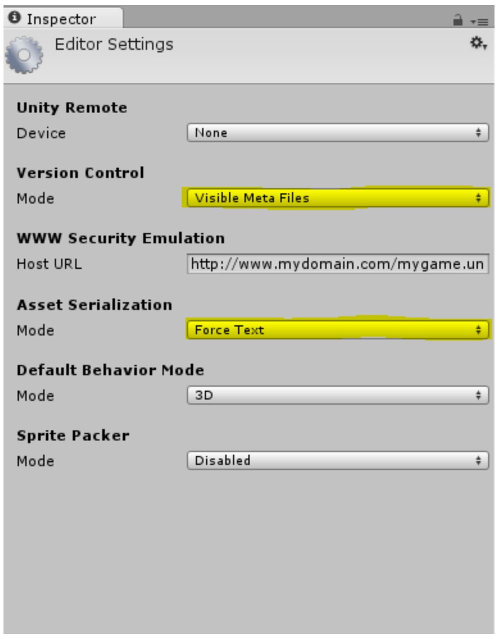
   git submodule update --init

10. In order to avoid uploading unneeded files, add the following contents to the .gitignore:

```
###########################
# SeniorLudens .gitignore
# v1.0
############################

### Unity ### [Ll]ibrary/ [Tt]emp/ [Oo]bj/

# Autogenerated VS/MD solution and project files
*.csproj*
*.unityproj*
*.sln*
*.suo*
*.user*
*.userprefs*
*.pidb*
*.booproj*

#Unity3D Generated File On Crash Reports sysinfo.txt

# Other generated files bin/
obj/


### Blender####
*.blend?
*.blend?.meta

### Distribution files ####
dist/
Docs/

### Testing ###
coverage/
TestResult.xml
```

If you prefer you can download it from: SeniorLudens .gitignore

11. Modify the editor project settings Unity: Edit -> Project Settings -> Editor to use text serialization and visible metafiles, as show in the next image:



Inspector

Editor Settings

Unity Remote

Device: None

Version Control

Mode: Visible Meta Files

WWW Security Emulation

Host URL: http://www.mydomain.com/mygame.un

Asset Serialization

Mode: Force Text

Default Behavior Mode

Mode: 3D

Sprite Packer

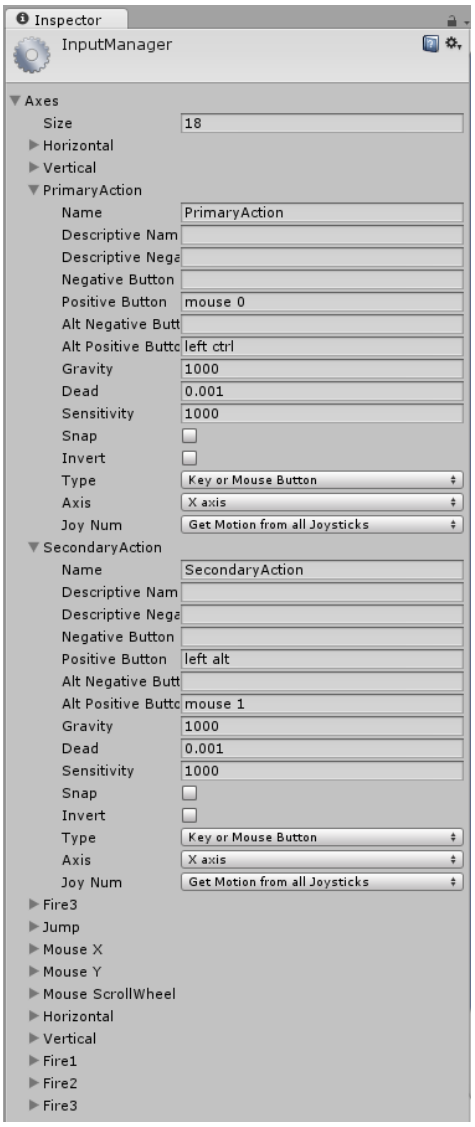
Mode: Disabled

Figure 1 Enable Visible Metafiles

12. In Unity, define the SeniorLudens settings:

    - Edit -> Project Settings -> Input -> in the Inspector panel -> Axes:

      Set parameter *Fire1* to *PrimaryAction*
      Exchange the values of parameter *Positive Button* and *Alt Positive Button*
      Set parameter *Fire2* to *SecondaryAction*


Inspector
InputManager

```
▼ Axes
    Size                 18
  ▶ Horizontal
  ▶ Vertical
  ▼ PrimaryAction
      Name               PrimaryAction
      Descriptive Nam
      Descriptive Nega
      Negative Button
      Positive Button    mouse 0
      Alt Negative Butt
      Alt Positive Buttc left ctrl
      Gravity            1000
      Dead               0.001
      Sensitivity        1000
      Snap               ☐
      Invert             ☐
      Type               Key or Mouse Button
      Axis               X axis
      Joy Num            Get Motion from all Joysticks
  ▼ SecondaryAction
      Name               SecondaryAction
      Descriptive Nam
      Descriptive Nega
      Negative Button
      Positive Button    left alt
      Alt Negative Butt
      Alt Positive Buttc mouse 1
      Gravity            1000
      Dead               0.001
      Sensitivity        1000
      Snap               ☐
      Invert             ☐
      Type               Key or Mouse Button
      Axis               X axis
      Joy Num            Get Motion from all Joysticks
  ▶ Fire3
  ▶ Jump
  ▶ Mouse X
  ▶ Mouse Y
  ▶ Mouse ScrollWheel
  ▶ Horizontal
  ▶ Vertical
  ▶ Fire1
  ▶ Fire2
  ▶ Fire3
```

Figure 2 Set default actions

13. Do not close your Unity3D project, just proceed to the next step to create the basic SeniorLudens project files . Remember that from now on, you should periodically, add your project files, make a commit and push it to your repository:

    - *git add (whatever needed, check with git status)*
    - *git commit*
    - *git push*

1.2.2.2- SeniorLudens project files

A SeniorLudens project needs the following files:

- **myprojectConfiguration.cs** that contains the definition of the actions that are specific actions of your world.
- **myprojectWorld.xml** that defines all the objects of your world

- **scenarioConfiguration.xml** that describes the initial content of all the scenes of your world. [1]
- For each task:

  mytask.xml that describes the contents of a task. [1]

1. Create the C# file *MyProjectConfiguration.cs*. By now, just copy the file enclosed below. Do not forget to substitute *MyProject* by the name of your project.

```
using SeniorLudens.Unity;

public class MyProjectConfiguration :
SeniorLudensUnityAppConfiguration {

        public override void Configure
        (SeniorLudens.Core.Application app)
        {
                // Configure your application here adding
                your custom behaviou

        }

}
```

2. Create the xml file *MyProjectWorld.xml* as shown below. As you can see it is almost empty. It only includes a default object called *room*. With this you have enough right now, so just copy it as it is. Later, you will add the definition of all the objects of your world after this definition.

```
<?xml version="1.0" encoding="UTF-8" ?>
<world>

<objectdef id="room">
   <states default="decoration">
   <state name="decoration"></state>
   </states>
</objectdef>
```

3. Create the xml file *scenarioConfiguration1.xml*. By now, this file is also almost empty. Within the tags <positions> </positions> you'll put the initial positions of the instances of the objects existing at the beginning of the game in each scene of the world.

```
<?xml version="1.0" encoding="UTF-8" ?>
<scenarioconfig>

</scenarioconfig>
```

4. Create a first task FirstDemoTask.xml that simply closes the game after a lapse of time. You can copy the one below. Try to understand it. It has only the introduction step and one track composed of a sequence of three blocks. The first block is a conditional block that requires waiting 60 seconds. The second block is a system action consisting of showing the message ‘Bye!’. Finally, the third block is again a condition of a waiting time of 20 seconds.

```
<?xml version="1.0" encoding="UTF-8" ?>
<task>
    <meta>
        <name>Test task</name>
    </meta>
    <introduction>
       <track>
             <sequence>

                <condition>
                   <expression>
                         <wait>60</wait>
                   </expression>
                   </condition>

                   <action>
                         <subject>system</subject><verb>setTextPro
                   perty</verb><directobject>message</directobject
                   >
                         <param name="text">Bye!</param>
                   </action>

                   <condition>
                   <expression>
                    <wait>2</wait>
              </expressi
              on>
                <condition>

            </sequence>
        </track>
    </introduction>
</task>
```

5. In the next step you will create a warehouse for your world.

1.2.2.3- Create a new warehouse

In order to create a new warehouse, you just need to create the directory *warehouseMyWorld* in *Assets*. This warehouse will contain the objects that are specific to your world, have been designed for it and are not shared by any other world. In your world, you will use the objects of this warehouse and those of the SeniorLudens warehouse that are common to all SeniorLudens projects. See warehouses to know more about the structure of the warehouses in SeniorLudens.

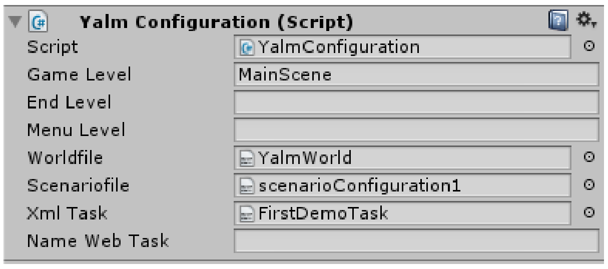
1.2.2.4- Define the world

You've just done the *Basic setting for the creation of a new world* of your world. Unity 3D provides a default scene. Use it to create a new component that represents your world.

6. In the *Assets* directory create a new directory called *Scenes.* This is where you will store your scenes.
   - Save the default scene: *File* -> *Save Scene* (choose a convenient name and indicate the newly created direc- tory *Scenes*)
7. Create an empty object. Call it after your world's name (herein

   MyWorld). *GameObject* -> *Create Empty* (name it as *MyWorld*)
8. With the empty object selected, add the component *MyWorld Configuration* to it. It is the script that you've created in Step *SeniorLudens project files* :

   In the *Inspector* panel -> *Add Component* -> *MyWorld*
9. Configure *MyWorld Configuration* in the corresponding fields of the components panel. The Game level is the name of the scene. The world file is the xml file of definition of your world (*NameWorld.xml*), the scenariofile *ScenarioConfiguration1.xml* and the task *FirstDemoTask.xml*. See the image below for the world called Yalm.



Yalm Configuration (Script)

Script	YalmConfiguration
Game Level	MainScene
End Level	
Menu Level	
Worldfile	YalmWorld
Scenariofile	scenarioConfiguration1
Xml Task	FirstDemoTask
Name Web Task	

Figure 3 Configure MyWorld Configuration

1. Create a *prefab* with the empty object by dragging it from the *hierarchy panel* to the assets directory in the Project panel at the lower part of the screen. A prefab is a Unity3D template of the objects and its components.
2. You are now ready to work on this scene add to it structural elements and objects. We are now ready to create a scene.

1.2.2.5- Create a scene

Connect the scene with *SeniorLudens GameKit*

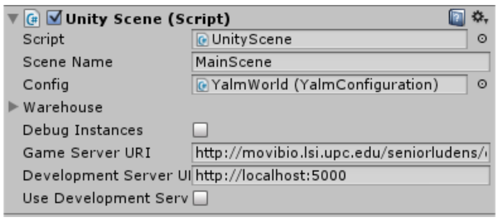
1. If you've just defined a new world (*Define the world* section) open it. You have already created a new scene. Otherwise, create a new scene in your old world and save.
2. In *Unity 3D* create an empty object that will represent the scene. Name the empty after the scene's name:

   GameObject → Create Empty → (name it as MySceneApp)

3. Assign the script *SeniorLudens Scene* to recently created object *MySceneApp*:

   Window → Inspector → Add Component → SeniorLudens →SeniorLudens Scene

4. Fill the corresponding fields: the configuration script is MyWorlConfiguration.cs, the **Game Server URI** is http://movibio.lsi.upc.edu/seniorludens/dev and the **Development Server** http://localhost:5000. See the image below where the name of the project is *Yalm*.



Unity Scene (Script)	
Script	UnityScene
Scene Name	MainScene
Config	YalmWorld (YalmConfiguration)
Warehouse	
Debug Instances	☐
Game Server URI	http://movibio.lsi.upc.edu/seniorludens/
Development Server Ul	http://localhost:5000
Use Development Serv	☐

Figure 4 Complete Scene configuration

5. Ensure that the *game level* parameter of your world prefab is the scene's name.

1.2.2.5.1- Create a Skybox

A skybox is a panoramic texture that represents the background of the scene: a sky or something similar. To create it, just follow the instructions of Unity3D Skyboxes[1]. Summarizing:

1. Open the render settings: *Edit → Render Settings*
2. In the *Inspector* window, on the default layout at the right side, press the *Skybox Material* button and search the material (e.g.: *sunny1 Skybox*).

Note: You can design skyboxes of your own or use those provided in the corresponding Unity3D package *Assets → Import Package → Skyboxes*)

---

[1] http://docs.unity3d.com/Manual/HOWTO-UseSkybox.html

1.2.2.5.2- Add lights

To start, add just a basic Unity3D light[2], the available lights can be found on the menu under *GameObject → Light* or with the keyboard Alt-g Alt-l

For example to add a directional light you will select on the menu *GameObject → Light → Directional Light*' or with the keyboard *Alt-g Alt-l Alt-d*.

1.2.2.5.3- Create your basic structure (ground and walls)

You can create them directly in Unity3D, or alternatively create them with a digital content creation editor (e.g. Blender) and insert them in your project. In both cases, you must be aware of the structure of the files and directory in SeniorLudens projects. Please read first the warehouses Section. You will also need to create the definition files of the objects.

The ground and walls are not likely to be shared by any other project and thus, in general, they will not be stored in the SeniorLudens warehouse. Thus, you will create them and store them in your project's warehouse in the corresponding directories as follows:

- In your project's warehouse, create a directory called *objects*, and inside it create a directory called *structure*. If your ground is an open-air landscape, create a directory called *landscapes*. In these directories, create a directory for each of your objects (e.g. in the *structure* directory create *walls*, *roof* and *ground*).
- Create the structure objects with *Unity 3D* ...

  In the menu *GameObject* -> *3D Object* –> select a plane, a terrain or whatever needed. In the *Inspector* panel, modify dimensions, position and other attributes.

- ... or import them from Blender

  – Read *first* the guidelines on how to create a Blender object to include it in a Senior-Ludens project in Section blender_objects. This will spare you a lot of time!

  – Save the *.blend* in the newly created directory.

  – In *Unity 3D*, drag the object from the *Project* to *Scene* and modify its property in *Inspector* panel.

After that, in both cases perform the following steps following the steps described in Section Unity_objects:

- Add the *VisualObject* component
- Add the *Collider* component
- Mark *Is Trigger*
- In the *Inspector* panel mark the *static* flag
- Create an *.xml* file with the definition of the object.
- Create the *prefab*

[2] http://docs.unity3d.com/Manual/class-Light.html

1.2.2.5.4- Add the definitions of the structural objects in the file ProjectWorld.xml

In the file YourProjectNameWorld.xml that you have created in the previous steps, add the definitions of the structural objects of the scene. For example:

```
      <?xml version="1.0" encoding="UTF-8" ?>
      <world>

      <objectdef id="ground">
              <states default="decoration">
                      <state name="decoration">
                                      <action     name="navigate"
                                      function="place"/>
                      </state>
              </states>
      </objectdef>

      <objectdef id="wall">
              <states default="decoration">
                      <state name="decoration">
                                      <action     name="navigate"
                                      function="place"/>
                      </state>
              </states>
      </objectdef>

</world>
```

1.2.2.5.5- Register the scene

Just skip this step by now until *SeniorLudens* supports multiscenes.

1.2.2.5.6- Add objects

To add objects in your scene, follow Section *Add an object to a world* section.

1.2.2.6- Create the user's avatar

SeniorLudens games are *first-person perspective*. This means that the environment is rendered from the point of view of the player's avatar. This avatar is a special SeniorLudens object composed of a camera and a very simple bounding box body used to control collisions. The avatar's camera is used at each frame to render the scene.

6. Add the user's avatar to the scene:

   Look for the object *avatar* in the SeniorLudens warehouse. Pick it in the *Project* panel and drag the *avatar* prefab to the *Scene*. In the *Inpector* panel, adjust position and rotation.

7. Remove the object *Main Camera*. You don't need it anymore, because you will use the

avatar’s camera.

8. Check that you can observe the scene from the avatar’s perspective:

   Press *Play* and move the mouse. You should be able to rotate the view direction.

   You’ll probably get various error messages in the *Console* panel. Don’t care about them, you’ll fix them in the next steps.

9. Add the definitions of the avatar to the project definition file **myprojectWorld.xml** (see *SeniorLudens project files*). Since the avatar is a compound of objects, you must add the definition of each of the parts that you want to be able to refer to in the game logics. In general, it is enough to define the *GUIcamera* that composes the avatar’s main camera and the avatar itself. Thus, add the following peace of definitions to the project definition file.

```
<objectdef id="GUICamera">
        <states default="decoration">
                <state name="decoration" model="GUICamera"/>
        </states>
</objectdef>

<objectdef id="user">
        <states default="motionless">
                  <action name="quit" function="subject"/>
                </state>
                <state name="motion">
                  <action name="stopNavigation"
                  function="subject"/>
                </state>
                <state name="staticCamera">
                  <action name="lockCamera" function="subject"/>
                </state>
        </states>
</objectdef>
```

10. Setup the navigation:

   (a) Check that all the objects that won’t move during the game (e.g the ground and the walls) are *static*.

      With the object selected, check *static* in the *Inspector* panel (at top right).

   (b) For all *static* objects, define if the *avatar* will be allowed to navigate automatically towards it or not.

      With the static object selected, in the *Window* submenu of the main menu, select the *Navigation* option. In the *Navigation* panel, select the *Navigation layer*. Choose *Default* if you want to allow navigation towards the object and *Not walkable* otherwise. For instance, with the object *ground* selected, choose: *Navigation Layer* -> *Default*.

   (c) At the bottom right of the *Navigation* tab, press the button *Bake*.

You'll be able to see in blue the navigation mesh[3].

(d) Adjust the mesh by tuning the parameters:

- *Radius*: the proximity margin to the obstacles. Be careful with this value: if it is too large, you won't be allowed to pass through narrow doors. A good value according to SeniorLudens scenarios scale is 0.2
- *Height*: a good value is 1.8. If it is too high, the avatar won't be able to pass through doors.

11. Verify that the avatar navigates and recognizes the different scene's objects.

    Press *Play* and click on the object you want to go to. On the bottom of the screen a message will indicate where you are navigating to.

Warning: If it doesn't work, check that the object you want to go to has a collider. In general a *Box Collider* is enough. However, if the object is a closed space (a room, for instance) and the avatar is inside, you need to put a box collider per wall or a *Mesh collider*. Check also that the flag *IsTrigger* is on.

1.2.2.6.1- Configure the avatar

Select the avatar and in the *Inspector* panel, in the *Nav Mesh Agent* component, you can modify interesting features:

1. **Speed:** navigation speed [p.ex: 1]
2. **Stopping Distance:** Distance to the clicked object at which the avatar stops

Note: Apparently, the minimum *Stopping Distance* is 0.8. Below, the behaviour is as if it was 0.

Warning: Choose carefully the distances, because if they are too large, the objects may all fall out of the scope of the avatar.

1.2.2.6.2- Configure the avatar's camera parameters

Select *Main Camera* in the *avatar* compound,. In the panel *Inspector* modify the component *Camera*, for instance, the clipping planes and the type of projection.

[3] http://docs.unity3d.com/Manual/nav-InnerWorkings.html

1.3- Modify a world

1.3.1- Before you start

1.3.1.1- Who is able to modify a world and why?

Your organization is provided with a world composed of a set of objects and a set of scenes (scenario). Using the Scenario Editor, you are able to create graphically configurations of the *scenes* based on existing objects of the world. Using the *Task Editor*, you are able to define tasks in a specific configuration of your world.

If you are missing objects or behaviours in your world, you will need to ask for them to the SeniorLudens management board. They will create the objects, program the new behaviours and include them in your world.

In this document, we explain how to modify a world adding new objects and new functionalities. You will only be able to this task if you have access to the SeniorLudens Game Kit. You must be familiar with Unity3D to do it. Skip this document if you are simply willing to modify a configuration of the scenario and use the *Scenario Editor* instead.

If you need to create a new world from scratch continue reading this section and then jump to Section *Create a new world*.

1.3.1.2- Learn about the existing resources

The objects available in your world come from two repositories:

- the SeniorLudens repository, available in all the SL projects.
- your own project repository.
-

These repositories are in the Assets directory: warehouseSL and warehouseProject. Take a look inside and look the available resources. They are classified according to their category: food, accessories, construction etc.

Note: May be you already have what you need!! See SeniorLudens Warehouses section to know more about the structure of these repositories.

Whether you create new objects or modify them you need to know the concepts of SeniorLudens object, Visual Object, Unity Objects and Prefab.

1.3.2- Add an object to a world

In this section you will learn how to add an existing object stored in a warehouse to your world. If you want to create a new object from scratch, go to *Create a new object* section.

Adding an object to a world will allow users of the Scenario Editor to create instances of that object for different scene configurations.

There are two ways of adding an object to a world: either you add directly an instance of the object in a scene of the world, or you add the object to the world without creating any instance of it. In the latter case the object will not belong to the initial configuration of the scene, but it will be available in the Scenario Editor to create instances of it in other configurations. Keep in mind that in the Scenario Editor, you will only be able to manage objects that are in the world.

The core of SeniorLudens Game Kit implements a lot of actions such as to pick, to drop and to change state. However, some objects have very specific actions or autonomous behaviour that are not included in the core. These objects have they own scripts. When adding them in a world, you will need to add also their actions. Follow the instructions to do it.

To add an instance of an object to the scene:

1. Drag the prefab from the warehouse.

To add a new object to a world without adding an instance of it in a scene, follow the steps:

1. Open the scene of your world in which you are working
2. Select the empty Unity object that represents a scene of your world (*MainScene*, e.g.)
3. In the left-side panel, add a new entry to the *Warehouse* drop-down submenu: *increase the number of objects* drag the prefab of the new object to the new entry.

In both cases:

1. In the world definition xml file add the description of the new object (see *SeniorLudens project files* subsection). The description is available in the warehouse in the file *definition.xml* just open it and cut and paste in *MyWorld.xml*.

   Check if the object has update scripts to implement autonomous behaviours not included in the core. If so, they should be in a folder called *scripts*, and within it in a subfolder *cs*. They should be named after the object's name. This is the case, for instance of the objects *extractor*. Its update script reproduces the sound of an extractor when it is on. The script is called *extractor.cs*. Take a look at the scripts and check if they are defined on the Visual Object or on the object. In the former case, the prefab already includes the behaviour programmed in the script, so you don't have to do anything. This is the case of the thermometer has a *thermometer.cs* script, but defined on the *ThermometreVisual*. In the latter case, you should notify the existence of the behaviour to the world. For that, open the script *Configure* of MyWorld class definition, and add the sentence *app.AddType(MyObjectTypeID, typeof(MyObjectType)*. This is the case of the extractor. For instance:

```
public void Configure (SeniorLudens.Core.Application app)   {

    app.AddType("Extractor", typeof(Extractor));

}
```

2. Check if the object has action scripts to implement specific reactive actions. If so, they should also be in the folder *cs* of the folder *scripts*. They should be named after the action's name. For instance the object *egg* has the action *break* implemented in the script *BreakAction*. You must notify the world of the existence of this action. Open the script *Configure* of MyWorld class definition, and add the new Action (*app.AddAction(new MyActionName)*). For instance:

```
public void Configure (SeniorLudens.Core.Application app)   {

    app.AddAction(new SeniorLudens.Warehouse.Egg.BreakAction());

}
```

1.3.3- Modify existing objects

You can modify existing objects of your world by adding to them new visual styles, new states and new actions.

1.3.3.1- Create a new visual style for an object

Your world has a ball, looking as a football ball with black and white patches. You need a different style, with red and blue patches. The first thing that you need to think about is if the second ball is only a variation of the first one (a different visual style) or if it is actually a different object. For instance rugby balls and football balls may be different objects, because they may play a different role in a training game. If you need to create a different object proceed to Section *Create a new object*. Otherwise, continue in this section.

Objects have different visual styles: colours textures and graphical design. In order to add a new graphical style you need to know how to use and create materials and how to use and create textures.

1. First, check if you are creating a new style that didn't exist previously in your world or if you are simply creating an existing style for your object. In the former case, thing carefully the name of your style. Choose it in a way that it will be clearly identifiable. In the second case, use the existing style name.

2. Create the new 3D model that will represent the object's new style. A new style can be a change of material and texture or even a completely new mesh.

3. Save the new model in the folder *styles* following the schema warehouses.

4. If you are using Blender, do not forget to unpack the textures and store them in the folder *textures* in the object's folder. If you are using another modeler, separate the textures in a similar way.

5. Add the new style in the object's definition file *definition.xml*.

```
                <states>
                </states>

                <style name="kumato"> <!-- kumato will be the id of the
                style -->

                        <name>Kumato</name> <!-- Style name -->
                        <description>Tomato          variant          named
                        Kumato</description>

                        <!-- Definition of the style's tag:
                                 - fruit: because a tomato is a fruit
                                 - red: the color of the tomato.
                        -->
                        <tags>fruit, red</tags>

         <!-- For texture changes: -->
                        <visual state="STATE_NAME">
                                <texture>TEXTURE_IMAGE</texture>

                        </visual>
                    <!-- For mesh changes: -->
                        <visual state="NOM_DE_L'ESTAT">
                                <model>NOM_DEL_.BLEND</model>
                        </visual>

                </style>
```

6. Import the object in the *Unity3D* scene and create the *prefab.*

1.3.3.2- Create a new state of an object

Suppose that the tomatoes of your world have only one state: *full_raw*. You need to implement the action *cut* and add the state *half_cut_raw* to the tomato if you want to design tasks requiring to cut tomatoes. Let's do it.

7. Create a new state. It can include:
    - Texture modification: either in your object's modeler or directly in Unity3D.
    - Mesh modification: create a new 3D model with you 3D modeller.
    - New animation: create the animation with your 3D modeler.
8. Create a prefab for every state
9. Modify the object's definition_file and the *Add the definitions of the structural objects in the file ProjectWorld.xml* section. For instance, the

following definition file includes two states for a door object:

```
<objectdef id="door">
<states default="closed">
        <state name="closed">
                <action name="open">
                        <param name="animation">open</param>
                        <param name="state">opened</param>
                </action>

                <action name="navigate" function="place"/>
        </state>
        <state name="opened">
                <action name="close">
                        <param name="animation">open</param>
                        <param name="state">closed</param>
                        <param                     name="reversed"
                        type="bool">true</param>
                </action>

                <action name="navigate" function="place"/>
        </state>
</states>
</objectdef>
```

1.3.3.3- Add an existing behaviour to an object

Object's behaviour depends on the actions that are enabled on them. In this section, you will learn how to assign an existing action to a new object. For instance, add the action of disintegration to another existing object.

1. Open the file *definition.xml* of an object that already has the desired behaviour and copy the definition of the desired action.
2. Open the file *definition.xml* of the object to which you want to add the action. Paste the code to every state that you want to have these actions. For instance:

```
          <state name="full" model="apple">

                  <!--  for  the  state  'full'  we  already  had  the  actions
                  navigate, pick an

                  <action name="navigate" function="place"/>
                 <action name="pick" function="directobject"/>
                 <action name="drop" function="directobject"/>

                 <!-- and we now add the action destroy -->

                 <action name="destroy" function="directobject"/>

         </state>
```

3. Check the definition of the parameters.

1.3.3.4- Create a new action for an object

If SeniorLudens GameKit does not include any action that matches your needs, you'll need to implement the new action and assign it to your object.

1. Create the script of the action in C# for Unity3D and store it in the Scripts/cs folder of the object. Use the SeniorLudens API.
2. Add the new action to the definition of the object in the object definition file *definition.xml* as well as in the project definition file *MyProject.xml*.

1.3.4- Remove objects from your world

In general, objects should not be removed from a world unless they are erroneous. You never know if you may need them in a future scene!

If you really want to remove an object from your world, make sure that there are no instances of it in any of the world's scene. Then, remove its definition from the cml world definition file and, if it has actions and specific behaviour, remove them from the Configure method of the world configuration class.

1.3.5- Create a new object

You can create new objects either from scratch or by taking as a reference existing objects. The latter way is faster and convenient whenever you need to create new objects of a same family with different aspect and name but with similar behaviour, for instance an apple having already a pear.

1.3.5.1- Create a different object similar to an existing one

Now, imagine that you need to create a rugby ball as a different object from the already existing football, or, your world has tomatoes, but you need peppers. A pepper is essentially the same object as a tomato, it has the same actions (to pick, to drop...) but with different name and aspect. Let's create an object using another as reference.

1. Create the new 3D model
2. Unpack the textures that are needed for the new object and save them in the folder *textures* within the object's folder.
3. Copy the file *definition.xml* of the object that is similar to yours and modify it to adapt it to your object. Save the file in the object's folder.
4. Import the object to *Unity3D*:
    - As soon as *Unity3D* will open, the folder *Materials* will be created that contains the materials that the object uses.
    - Save it as a .prefab in the folder of the new object.

3.4.1 Create a new kind of object

1. Check that the type of object that you need is not currently provided by SeniorLudens GameKit. Recall that several different objects can be of the same type. For instance, there may be different objects of type “door” or “chair”.
2. Create a suitable name to the new type of object. Try to be precise, thinking that even now your object is the first of a category, may be you'll add others of the same category later on, so don't call it after its category but after its intrinsic nature. For instance, if you put the name “fruit” to an apple, then you will not be able to distinguish between fruits. Call it apple.
3. Add to the object definition file all the states and attributes it has. Modify accordingly the *MyProjectWorld.xml* file. For instance:

```
          <objectdef id="OBJECT">

          <states default="DEFAULT_STATE">

                  <state name="STATE_1">
                          <action                           name="ACTION_1"
                          function="FUNCTION_ON_SENTENCE"/>
                          <action                           name="ACTION_2"
                          function="FUNCTION_ON_SENTENCE"/>
                          <action />
                  </state>

                  <state name="STATE_2">
                          <action                           name="ACTION_1"
                          function="FUNCTION_ON_SENTENCE"/>
                          <action/>
                  </state>


          </states>

          </objectdef>
```

Add the description of the new object type to the documentation

1.3.6- Modify a scene

Use the *Scenario Editor* to create configurations of the scene: create, remove and move object instances of your world.

If you want to modify structural elements of the scene, that you cannot edit in the *Scenario Editor*, follow the steps of Section add new objects.

1.3.7- Add a new scene to your world

This section explains how to register new scenes. Currently, SeniorLudens allows you to have only one scene aside from the introduction and conclusion, so be aware that this documentation can change a lot.

1. First, create the scene. Do not forget to register it.
2. Then in the menu File -> Build Settings... -> drag the new scene from the Project panel to Scenes in build.

Note: If the scene is the beginning or the end of a task, you should add it to the *ProjectApp* component: With the object *ProjectApp* selected, in the *Inspector* panel, add the scene to the suitable component of *ProjectApp* (e.g. to *End Level* if it is an end-task scene)

1.4- Migrate a Unity Scenario

If you have already built a *Unity3D scenario* and you want to use it in SeniorLudens, it may be better to start first by creating a very simple world from scratch in order to understand the processes and concepts: *Create a new world* section. Once you'll have done this first experiment, you'll be ready to follow these instructions:

1. First install the platform: *Installation*
2. Apply the basic settings: *Basic setting for the creation of a new world*
3. Create the projects files: *SeniorLudens project files*
4. Add an empty world object and define the world: *Define the world*
5. Create an empty scene object and define it: *Create a scene*
6. Create a new warehouse for your project: warehouses.
7. Add the objects of your scenario to the warehouse: for every object, create a prefab (if needed), store it in the warehouse following SeniorLudens pattern and create the corre- sponding definition files: *Create a new object*.
8. For all the prefabs add the *VisualObject* component: Unity_objects.
9. Add the definition of all the objects in the world definition file.

2- Install the Scenario Editor

The Scenario Editor is available as a prefab at the SeniorLudens Warehouse, in order to include it in your world drag the prefab into your scene.

Now you have to configure some options of the scenario editor, as the scenario editor is build using a Canvas element you have to define the camera and the render mode; we recommend you to use the camera named GUI from the avatar and a distance of 80.

3- Task Editor

3.1.1- Reference site

The task editor is deployed temporarily in an intermediate server located at: http://selte.cbim.it/. It will be integrated inside the SeniorLudens Platform infrastructure. Although it is not deployed in the final server, it is connected with the SeniorLudens Storage Server that is deployed on it.

3.1.2- Introduction

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

Deploying the full state diagram of all possible user actions is very tedious and prone to errors. Therefore, the task editor tool will require trainers to define only the reference task, this is the correct way of doing things.

For the reason Task Editor Tool makes use of Blockly as Visual Editor that allows users to write flows by plugging blocks together.

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 bloc 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 if they are correct or no. 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.

3.1.3- Features

3.1.3.1- How include new blocks

You can find the existing set of blocks in the toolbox (Task blocks) as follow:

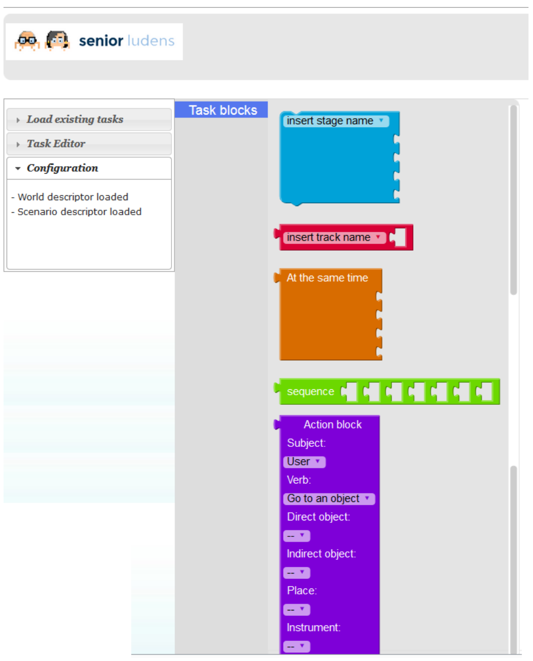


Figure 5 Toolbox which include complete set of blocks

The mandatory type blocks are as follow:

- Stage
- Track
- Sequence or At the same time
- Action

3.1.3.2- Workspace

Workspace is the section which include the blocks used to model the task

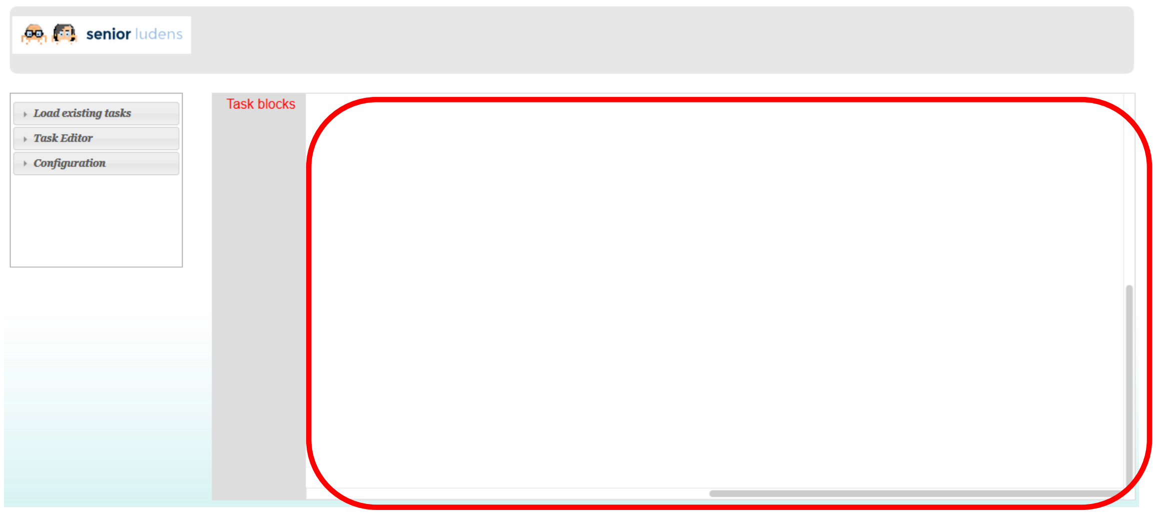


Figure 6 Task Editor working area

To inject a block into workspace is enough select the desired block from toolbox and drag it on workspace

3.1.3.3- Modify within the block

3.1.3.3.1- Duplicate

This feature provides to duplicate the workspace selected block.

3.1.3.3.2- Delete

This feature provides to remove the workspace selected block.

3.1.3.3.3- Run a contextual description of block

Example:
We can try to put into the workspace the Action Block and with right click of mouse on the block area, testing the functions as listed above:

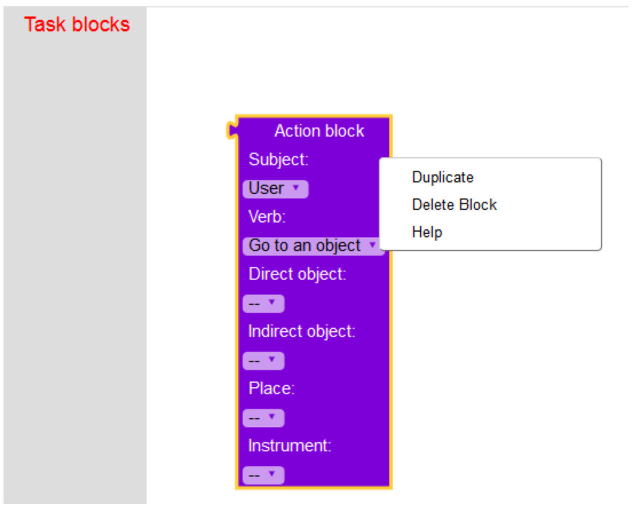


Figure 7 How modify an action block

Meanwhile, only for types Stage and Track, you can "rename" the title of the block.
For example, we can try to put into the workspace the Stage Block and with left click of mouse on the dropdown area, testing the function "New variable":

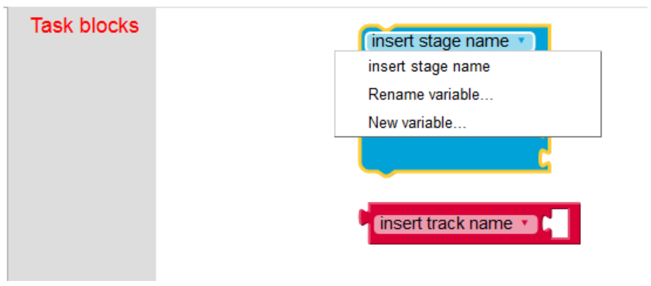


Figure 8 How modify stage/track block

3.1.3.4- Load existing task descriptor

You can load an existing task by clicking on “go to an object” within left menu (Load existing tasks) as follow:

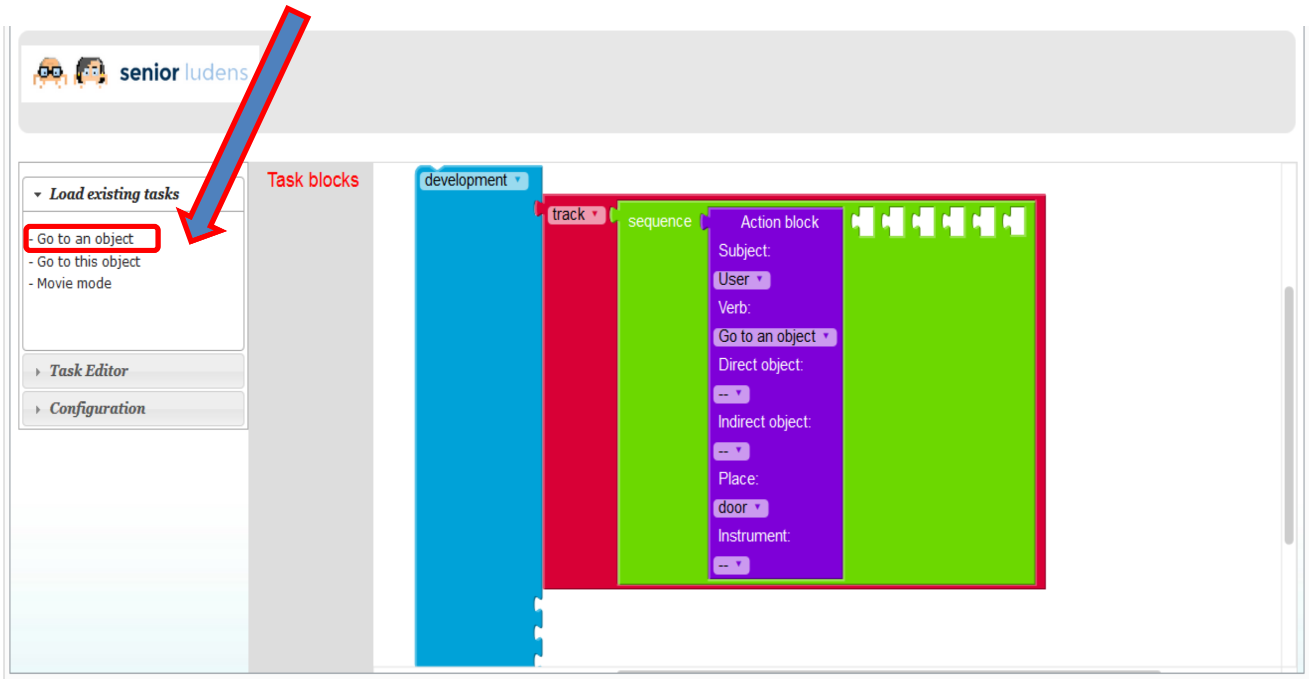


Figure 9 How load existing task

3.1.3.5- Create new task descriptor

After inserting or modified an existing task, you can create and show new task descriptor simply clicking “Creating the new TE descriptor” within the left menu (task editor) as follow:

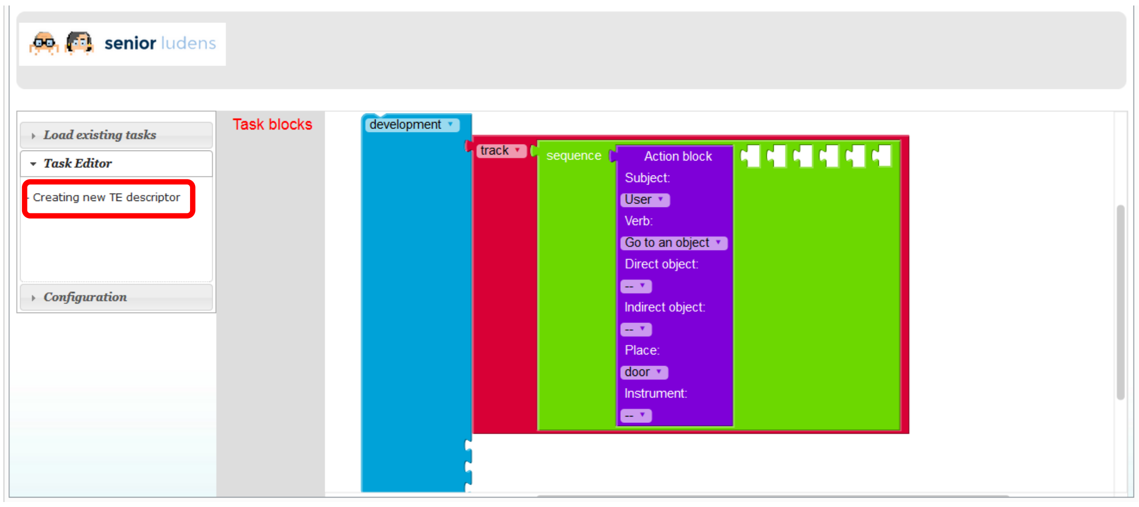
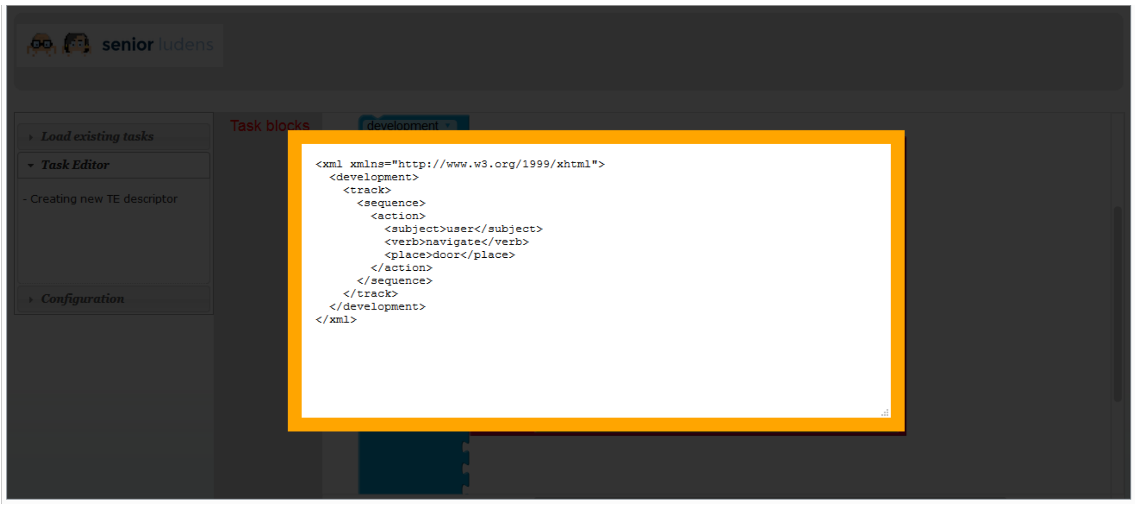


Figure 10 How create new TE descriptor



```
<xml xmlns="http://www.w3.org/1999/xhtml">
  <development>
    <track>
      <sequence>
        <action>
          <subject>user</subject>
          <verb>navigate</verb>
          <place>door</place>
        </action>
      </sequence>
    </track>
  </development>
</xml>
```

Figure 11 Visualization of the new TE descriptor XML

3.1.3.6- Put action modules in parallel

Task editor is able to manage the action block also in parallel to communicate to the Training Program Module how the action should be execute, at the same time or in sequence.

Example:

We can try to put a parallel block into a clean workspace and insert two actions block into this one as follow:

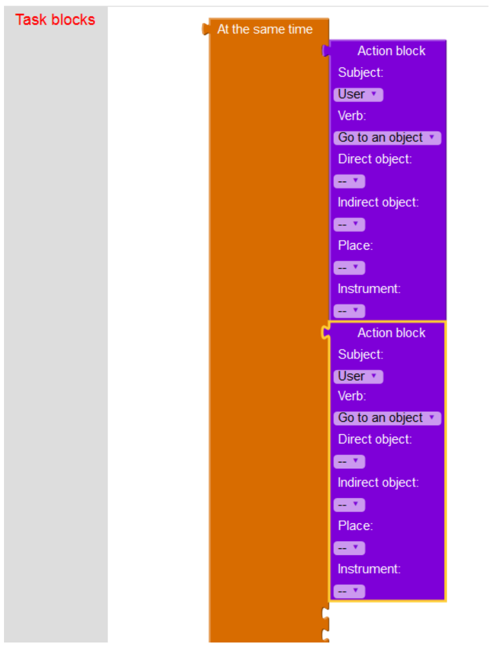


Figure 12 How manage the action blocks in parallel

4- Training Plan Editor

This module has not been developed in the first iteration of development inside of the SeniorLudens agile cycle which rules the work in the project. It will be built and integrated in the next phase.

Figures and tables

Acronyms

Acronym	Explanation
SL	SeniorLudens
SLGK	SeniorLudens Game Kit
SDK	Software Development Kit
TE	Task Editor