Let's make a game including competencies!

This document will show you how to use our Asset to create a game for assessing or even learning competencies. For starters, we will discuss some basic vocabulary used in this document, you can always come back here to look them up:

- **Competence:** A competence is the basic building block in our framework and describes a knowledge/skill independent of other competencies. This could for example be 'turning on the computer' and 'knowing the name/symbol of the email client on the desktop' and so on, when talking about the Area of 'How to write an email to John'.
- **Prerequisite Relation:** Two competencies in each case might be related to each other. Such a relation could have multiple appearances, we focus on a very special one: the prerequisite relation. If two competences are in such a relation, one is called the prerequisite competence, the other is the base competence. This means, that the base competence can only be mastered, if the prerequisite competence is mastered. For our example we could introduce a prerequisite relation between the base competence 'knowing the name/symbol of the email client on the desktop' and the prerequisite competence 'turning on the computer'. We would then assume, that someone who does not know how to turn on a computer could not know what the name/symbol of the email client on the desktop is.
- **Knowledge Domain:** This term describes an area of knowledge. In our above example this would be 'How to write an email to John'. Such a knowledge domain can be (and will be in our case) modelled with a set of competencies and their relations.
- **Domain Model:** A representation of the knowledge domain (there might be multiple different ones) including competencies and prerequisite relations in a computer readable XML format.
- **Domain Expert:** Someone who is an expert on the chosen knowledge domain. Such a person is normally able to easily assign competencies and their relations to a knowledge domain.
- **Competence Assessment:** This term describes the assessment of the player's competencies, which of the included competencies are already mastered and which are still there to learn.

We now start with our task: Creating a game including competencies! Therefore, we must choose a knowledge domain: what do we want to teach by playing this game? And then the associated competences need to be identified.

In these blue boxes an example of the just discussed content is given. We are using our example domain 'How to write an email to John'. We already got two competencies, and we add more, identified by the shortcut [C-]:

- [C1] Turning on the computer
- [C2] Knowing the name/symbol of the email client on the desktop
- [C3] Knowing how to open the email client on the desktop
- [C4] Knowing the login credentials
- [C5] Knowing John's email address
- [C6] Being able to write an email
- [C7] Knowing how to send an email to John in the client
- [C8] Knowing how to close the email client

- [C9] Turning off the computer

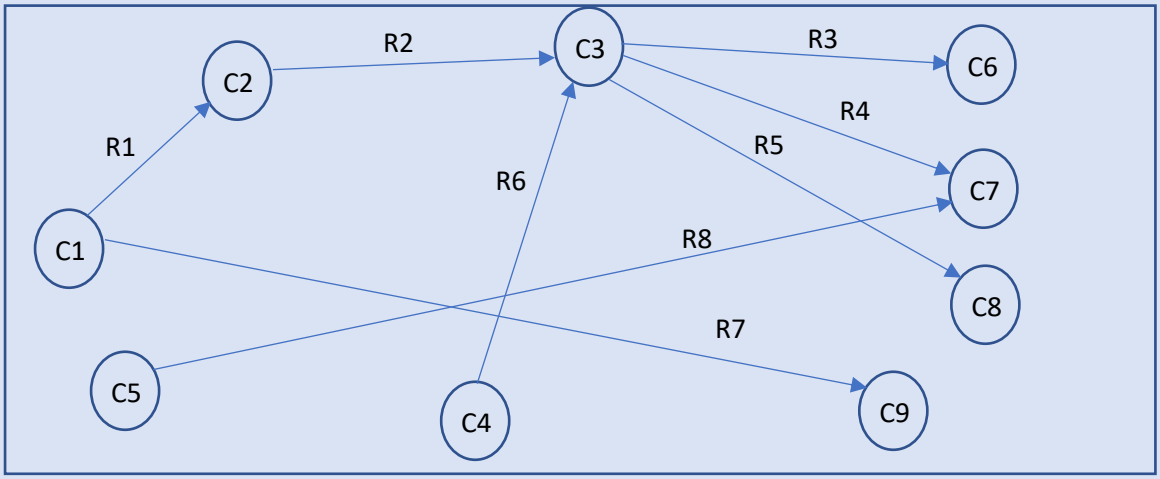
We now got the base set of competencies included in our knowledge domain. Of course, the included competencies depend on the concrete background, for example: the operating system, or the system setting/state ('Knowing John's email address' might be exchanged with 'Knowing how to load John's email from the address book').

After the competencies are known, the relations between them need to be identified.

In our example we go with the following relations (notation: 'prerequisite competence'->'base competence'), identified by the shortcut [R-]:

- [R1] 'Turning on the computer' -> 'Knowing the name/symbol of the email client on the desktop' [C1->C2]
- [R2] 'Knowing the name/symbol of the email client on the desktop' -> 'Knowing how to open the email client on the desktop' [C2->C3]
- [R3] 'Knowing how to open the email client on the desktop' -> 'Being able to write an email' [C3->C6]
- [R4] 'Knowing how to open the email client on the desktop' -> 'Knowing how to send an email to John in the client' [C3->C7]
- [R5] 'Knowing how to open the email client on the desktop' -> 'Knowing how to close the email client' [C3->C8]
- [R6] 'Knowing the login credentials' -> 'Knowing how to open the email client on the desktop' [C4->C3]
- [R7] 'Turning on the computer'' -> 'Turning off the computer' [C1->C9]
- [R8] 'Knowing John's email address' -> 'Knowing how to send an email to John in the client' [C5->C7]

Of course, these relations are strongly context dependent, for example R6 assumes that we need our login credentials to open the email client. This might not be true for all email clients. Another thing to discuss: One could argue, that 'turning on the computer' is a prerequisite to 'Knowing how to open the email client on the desktop'. This is true, but since we already got the relations 'Turning on the computer' -> 'Knowing the name/symbol of the email client on the desktop' and 'Knowing the name/symbol of the email client on the desktop' -> 'Knowing how to open the email client on the desktop' this also covers the relation 'turning on the computer' -> 'Knowing how to open the email client on the desktop'. A graphical representation is given here:



We now got all the required information for our purpose. The only thing left to do is to make this information accessible for the computer. Therefore, we use the data format XML and store the information in a file. This XML formatted information is then called a Domain Model.

A simplified domain including only the information discussed yet looks like the following:

```
<domainmodel>
 <elements>
  <competences>
   <competence id="C1"/>
   <competence id="C2"/>
   <competence id="C3"/>
   <competence id="C4"/>
   <competence id="C5"/>
   <competence id="C6"/>
   <competence id="C7"/>
   <competence id="C8"/>
  </competences>
 </elements>
 <relations>
  <competenceprerequisites>
   <competence id="C2">
    <prereqcompetence id="C1" />
   </competence>
   <competence id="C3">
    <prereqcompetence id="C2" />
    <prereqcompetence id="C4" />
   </competence>
   </competence>
   <competence id="C6">
    <prereqcompetence id="C3" />
   </competence>
   </competence>
   <competence id="C7">
    <prereqcompetence id="C3" />
    <prereqcompetence id="C5" />
   </competence>
   </competence>
   <competence id="C8">
    <prereqcompetence id="C3" />
   </competence>
   </competence>
   <competence id="C9">
    <prereqcompetence id="C1" />
   </competence>
  </competenceprerequisites>
</domainmodel>
```

Now we have the basic set up for doing competence assessment. Therefore, we assume that each competence is either mastered or not mastered. To decide to which category a competence belongs we are using probabilities assigned to each competence. These probabilities are manipulated in response to in game actions. A threshold separates mastered competencies (high probability value) from not mastered competencies (low probability value). There are multiple ways to update competencies, the simplest one is to directly update one competence.

When we want to update a single competence, we can use the following C# pseudo code, when ‘caa’ describes the instance of the competence assessment asset:

```
CompetenceAssessmentAsset.updateCompetenceState(“C3”, true);
```

Or

```
CompetenceAssessmentAsset.updateCompetenceState(“C3”, false);
```

In the first case, the competence is upgraded, in the second case it is downgraded. But, what happens, when a competence is updated (down- or upgrade)?

Therefore, the term related competence is important. These competencies are easily identified when following the arrows starting from the updated competence and only going one direction, i.e. either just in arrow direction or just against arrow direction. When updating a competence, we also update all related competencies, the closer the competence is located to the updated competence the stronger the update is.

When looking at the graphical representation of the competencies, we see, that the competencies C1, C2, C4, C6, C7, C8 are either prerequisites or base competencies to the competence C3. They are either in direct or indirect relation – competence C1 for example is in indirect relation (more than one arrow between C1 and C3) to competence C3 via the competence C2. All other competencies are in direct relation (one arrow between these competence and competence C3) to the competence C3. These competencies are easily identified when following the arrows starting from C3 and only going one direction, i.e. either just in arrow direction or just against arrow direction.

When updating C3, we also update the related competencies, the closer the competence is located to the updated competence the stronger the update is.