Impact Amplification Kit - PLEIADE Intellectual Output 6

“PLayful Environment for Inclusive leArning Design in Europe”

“PLEIADE”

Project No. 2020-1-IT02-KA201-080089

Co-funded by the Erasmus+ Programme of the European Union

Project ref. number	2020-1-IT02-KA201-080089
Project title	**PLEIADE — PLayful Environment for Inclusive leArning Design in Europe**
Document title	Impact amplification kit
Document Type	Intellectual Output (accompanying document)
Document version	1
Previous version(s)	–
Planned date of delivery	30/06/2023
Language	This accompanying document: English The kit: English, Italian, Bulgarian and Greek.
Dissemination level	Public
Number of pages	26
Partner responsible	CNR-ITD
Author(s)	Passarelli M., Pozzi, F., Persico, D., Volta, E.
With contributions by:	Translations provided by: Marilina Lonigro (Rocca-Bovio-Palumbo school), Christos Kyriakides (Neapolis Gymnasium), Anita Marietova (144 Narodni Buditeli school). Appendix 2 has been authored by Matteo Bicocchi, Pietro Polsinelli and Carlo Innocenti.
Revised by:	Marieta Angelova (144 Narodni Buditeli school), Elitsa Peltekova (UniSofia)
Abstract	This document accompanies the sixth Intellectual Output (IO6) of the Erasmus+ project “PLayful Environment for Inclusive leArning Design in Europe” (PLEIADE), which is the Impact Amplification Kit. According to the project proposal, the kit aims to amplify the impact of PLEIADE by providing stakeholders who are not

	directly involved in the project with the tools and support they need to implement project results in their own context. The kit consists of an online survey that guides users towards choosing the most suitable resources from those produced by PLEIADE. The kit is available in English, Italian, Bulgarian and Greek. The kit includes two appendices . Appendix 1 contains the results of the evaluation of the teachers’ professional development pathway carried out in PLEIADE. The evaluation is mainly based on the data collected during IO1, IO2 and IO3. Appendix 2 provides technical details for developers who want to customise or further develop IO2, i.e. the Hybrid I4Ts game. This game is one of the main outputs of PLEIADE
Keywords	amplification, dissemination, exploitation, project results, teachers’ professional development
DOI	https://doi.org/10.17471/54021
How to cite	Passarelli M., Pozzi, F., Persico, D., Volta, E. (2023). *Impact amplification kit* (PLEIADE Intellectual Output No. 6). https://doi.org/10.17471/54021

Table of Contents

1. Executive summary

This document accompanies the sixth Intellectual Output (IO6) of the Erasmus+ project “PLayful Environment for Inclusive leArning Design in Europe” (PLEIADE), the Impact Amplification Kit.

According to the project proposal, the kit aims to amplify the impact of the PLEIADE by providing stakeholders who are not directly involved in the project with the tools and support they need to implement project results in their own context. The kit consists of an online survey that guides users towards choosing the most suitable resources from those produced by PLEIADE. The kit is available in English, Italian, Bulgarian and Greek.

The kit includes two appendices.

Appendix 1 contains the results of the evaluation of the teachers’ professional development pathway carried out in PLEIADE. The evaluation is mainly based on the data collected during IO1, IO2 and IO3.

Appendix 2 provides technical details for developers who want to customise or further develop IO2, i.e., the Hybrid I4Ts game. This game is one of the main outputs of PLEIADE.

2. Introduction

This is the accompanying document of IO6, i.e. the project Amplification kit.

The kit itself is aimed to amplify the impact of the project, by providing guidance to any stakeholder (especially teachers, teacher trainers, school principals, researchers, policy makers, etc.) in selecting among the numerous resources produced within the project.

This accompanying document, instead, aims to describe the kit itself and provide the rationale behind its design and development. Moreover, the kit is also featured with two appendices: the first provides results of the evaluation of the main project outputs, while the second contains technical information for developers interested in taking up, customising, or further developing the Hybrid I4Ts game (IO2), which stands as one of the most innovative outcomes of the project.

To be noted, originally this output was expected to be led by EDEN, who was overall responsible for the project dissemination. Unfortunately, in Autumn 2022 EDEN withdrew from the project (see related contract amendment) and

the responsibility for IO6 was taken by CNR-ITD. The output was delivered in June 2023, according to the project time schedule.

This accompanying document is structured as follows: the next section contains a description of the amplification kit.

Appendix 1 contains the analysis of the data collected throughout the project with the aim to evaluate the main project results (IO1, IO2 and IO3). This appendix, unlike the rest of the document which is open and covered by a CC-BY-NC licence, is delivered as RESTRICTED.

Appendix 2 is a technical annex, complementing the information already contained in IO2 on the Hybrid I4Ts game, allowing further customization and thus transferability of this innovative outcome.

3. The Impact Amplification Kit

Diffusion of innovation on a large scale is a lengthy process in which early adopters play a key role to spearhead it in their professional environments and professional communities (Rogers, 1995). In order to enact systemic change, the energy and enthusiasm of early adopters has to be appropriately harnessed (White, 2007) by taking care of the barriers they encounter and nurturing their motivation in the challenging experience consisting of early change and innovation (Armstrong, 2019).

To this end, PLEIADE has developed Impact Amplification Kit specifically aimed at all stakeholders belonging to the early adopters and early majority cohorts. This tool is a key provision intended to help them to take the PLEIADE methods and tools up to their own contexts and in doing so innovate the practices of their communities on a peer-to-peer basis, which is recognized by several authors as the most effective approach for knowledge transfer in the teaching community (Vescio, Ross, & Adams, 2008).

The kit is self-explanatory, so that it can be used by teachers, school leaders, teacher educators, researchers or other stakeholders interested in adopting innovation concerning the PLEIADE methods and tools. The kit is expected to amplify and facilitate PLEIADE impact at European level.

The Impact Amplification Kit consists of an online survey that guides users toward choosing the most suitable resources produced by PLEIADE. The kit is accessible at this link:

https://survey.itd.cnr.it/123514

Figure 1. The kit welcome screen.

By answering questions aimed to understand the users’ interests and needs, users are guided within the set of numerous available resources that were produced during the project implementation.

The resources include not only the main project outputs (i.e. IO1, IO2, IO3, IO4 and IO5), but also all the teaching and learning materials produced and used especially during the teachers’ professional development pathway, along with the scientific papers produced, etc.

Overall, the kit includes more than 30 resources covering the main project topics, e.g. collaborative learning, social inclusion and education, learning design, gamification, practice sharing, and PLEIADE project itself. Most of the resources are available not only in English but also in other languages (Italian, Bulgarian and Greek). Moreover, most of the resources are available in multiple formats, e.g. slides, videos, texts, etc.

In the kit each resource is introduced by a brief description, so that the user can understand whether they fit with his/her actual interests and needs.

To make the kit more appealing, some of the resources are also introduced by short videos, aimed to give an idea of the resource itself at a glance.

4. References

Armstrong, E. J. (2019). Maximising motivators for technology-enhanced learning for further education teachers: Moving beyond the early adopters in a time of austerity. Research in Learning Technology, 27. https://doi.org/10.25304/rlt.v27.2032

Rogers, E. M. (1962). Diffusion of innovations (4th ed.). New York, NY, US: Free Press. http://dx.doi.org/10.2307/2573300

Vescio, V., Ross, D., & Adams, A. (2008). A review of research on the impact of professional learning communities on teaching practice and student learning. Teaching and Teacher Education, 24(1), 80-91. https://doi.org/10.1016/j.tate.2007.01.004

White, S. (2007). Critical success factors for e-learning and institutional change—some organisational perspectives on campus-wide e-learning. British Journal of Educational Technology, 38(5), 840-850. https://doi.org/10.1111/j.1467-8535.2007.00760.x

Appendix 1 - Evaluation of the PLEIADE Teachers’ Professional Development

This Appendix contains the evaluation of Teachers’ Professional development, encompassing IO1 (Blended Teachers' Professional Development pathway), IO2 (The Hybrid I4Ts Game) and IO3 (Gamified Platform for the Blended Training Activities).

Given that this document is delivered as RESTRICTED, it is contained in a separate document that can be accessed, upon request, from this link:

https://docs.google.com/document/d/11N4I8iP_-GOFaHJcNO63P2EuS19OkbjmrNLUvslsPsM/edit?usp=sharing

Appendix 2 - Developers’ Documentation for I4Ts Game

Introduction

This Appendix contains technical information about IO2, i.e. the Hybrid I4Ts Game.

In IO2 accompanying document, a User guide was provided, to support use of the game by any teacher/player.

As explained in IO2, some level of customization is already possible for the game at the user level: for example, the game is available in different languages (English, Bulgarian, Greek, and Italian) and can be even translated to other languages with no need of technical competences. Even the contents of the cards can be changed with no intervention by a developer.

In this Appendix, instead, we provide further technical details which can serve developers to take the game up and further customise, adapt, change it, according to needs different from those of the project. This is done to make the game even more transferable to other training contexts.

For completeness sake, we suggest you read this Appendix in conjunction with IO2, which contains the full description of the game.

Premises

The application is a client native application that connects online at startup and during service (http call) to a web based server. It has been built so as to be as stable as possible, compatible with generic hardware (Windows, Mac) and resilient to low definition printed markers. The client development tool (Unity) is a commercial product but is available with a full free version. The web service servicing the cards and their logic is built on a Prolog backend.

Full sources of all the modules composing the application (Unity project, C# code, Prolog backend, card service) are available through a public Git server here:

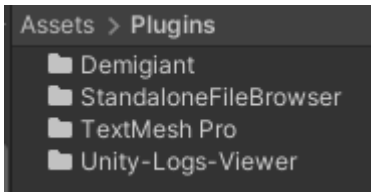
https://github.com/4Ts-game/4Ts-public-repository

allowing the customization and branching of the application.

Changes in the application client require to be proficient in Unity development, but do not require Prolog competence, as that is used as a service. Changes in the educational model behind the cards require Prolog competence to adapt the service to the intended model.

The **I4T Game client** is developed with **Unity3D** version 2019.4.9f1 (go to downloads at https://unity.com). The source code of the client is written in C#. It is realized under GPL license and it is open source.

The Unity plugins installed are all free, freely available and included in the distribution.



The Hybrid game is developed using the open source **“OpenCV” ArUco markers Library** (https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html) **ported for C#** and Unity (https://github.com/NormandErwan/ArucoUnity) by Erwan Normand.

Arcuro Markers Documentation:

https://medium.com/@calle_4729/using-mathematica-to-detect-aruco-markers-197410223f62

The Online Card service is developed in PHP, javascript, CSS and HTML.

It uses a Google sheet as a datasource converting it into a JSON object to retrieve and display the cards’ data.

Markers are produced based on the card ID using the **ArUco Marker generator** made by Oleg Kalachev (https://github.com/okalachev/arucogen).

Markers format: 4x4 (50, 100, 250, 1000)

Arcuro Markers online generator: https://chev.me/arucogen

All required downloads:

Unity: https://unity3d.com/get-unity/download/archive

Unity project sources: https://github.com/4Ts-game/4Ts-public-repository

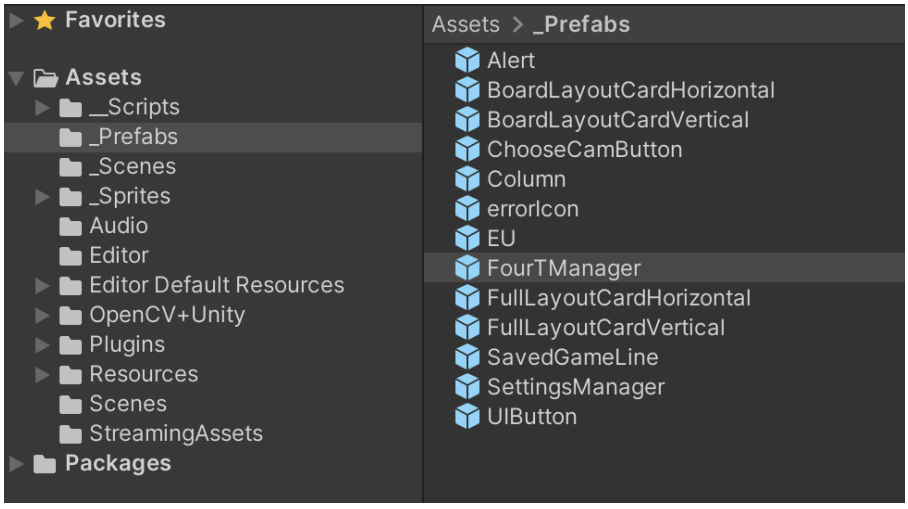
Prolog and its service: https://github.com/4Ts-game/4Ts-public-repository

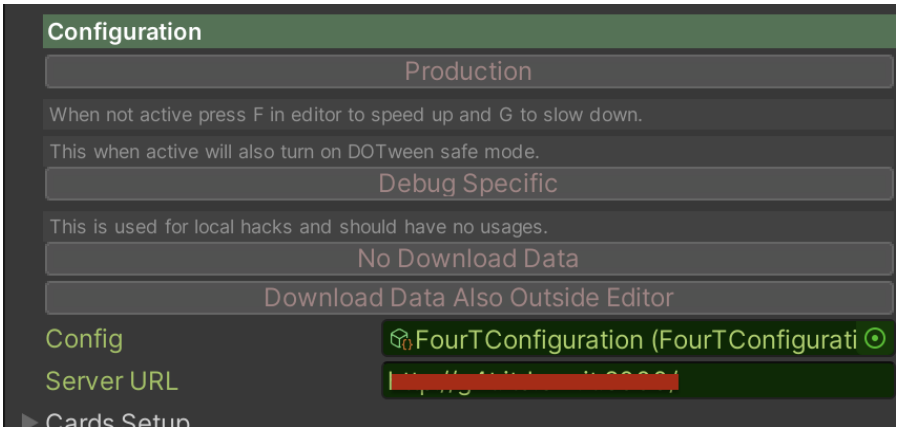
Card service: https://out.open-lab.com/pleiade

Configure the Unity Application

Prolog Server

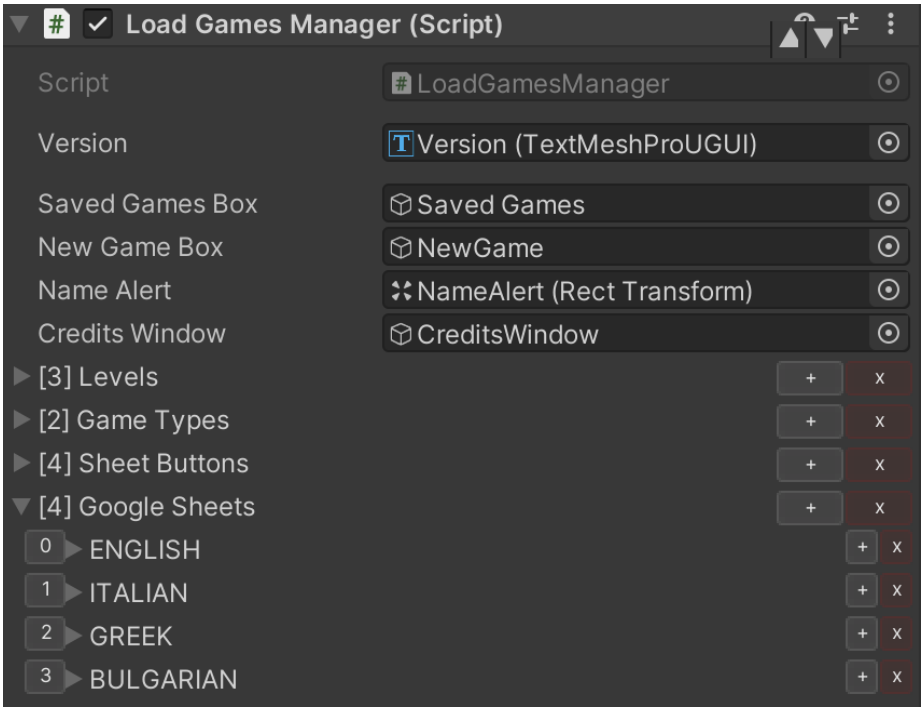
In order to configure the Prolog URL in the Unity application select and edit the FourTManager Prefab And modify the server URL option. Save the prefab.





Google Sheet Languages

In order to configure the Association of different languages and the corresponding Google Sheet, open the LoadGame Scene, select the LoadGameManager asset and add / change in the Inspector the Google Sheets option.



The flow of code and Unity scenes in the client application

In order to edit the Unity application you need to have the appropriate Unity version and a code editor (like Visual Studio as provided by the Unity download).

The application is composed of three scenes: Intro, LoadGames and Game. The application must start from the Intro scene.

The overall namespace of the classes is **FourT**. Inside the Unity project, as usual the classes are defined in the Assets/_Scripts folder.

Classes are in **bold** and methods in *italic*.

Intro scene

This is just a passthrough scene that presents a splash and loads the LoadGames scene.

LoadGames scene

When launching the application, in all cases the first class to set up is the (unique) instance of **FourTManager** which will persist across scenes.

This singleton class will load the cards either from an online spreadsheet on from the local Resources folder, and then will load persistent data of previous games if available. It also handles saving games through the **Persistence** class.

The main scene-specific manager is **LoadGamesManager**: this will set two properties that determine the game’s behaviour:

FourTManager.I().*Game.Level*:

Value 0: Visually 1, with full feedback on played cards.

Value 1: Visually 2, like 0 but does not check the technique cards.

Value 2: Visually 3, this does no checks.

FourTManager.I().*Game.GameType*:

Value 0: Purely digital game.

Value 1: Hybrid, integrated with a webcam detecting the board (corners and card locations) and cards.

Defined those, the GameScene scene is loaded.

The **Persistence** class: this class saves and loads games with a simple JSON serialization based on a built in service, **JsonConvert**.*SerializeObject*.

GameScene

The main scene manager of the game scene is **BoardManager**. This class designs the board (eventually loaded from persistence), setups a listener for clicks in card locations and launches the appropriate classes according to the game type.

In case the board is clicked (digital version) or a physical card is placed (hybrid), the card chooser panel is activated (method *ShowCardChooser*) and on a choice, the method *PlaceCardOnBoard* is called.

On every card played, if the game’s level is not “3” (2 internally) the completeness check is called:

ServerJob.I.*CompletenessCheckCall()*

And also the game state gets persisted with a call to (see the documentation below).

FourTManager.I().**Persistence**.*SaveGame()*;

More classes

Assets/_Scripts/FourTConfiguration

This is a typical Unity configuration class (ScriptableObject) that keeps references to graphical assets.

Assets/_Scripts/Model/Board.cs

Board navigation methods for reaching the played cards.

Assets/_Scripts/Model/Card.cs

Properties and methods for defining cards, their type, specific properties and runtime their position on the board. Also methods for getting the card full instance given their type and/or ID.

Assets/_Scripts/FourTMarkersManager.cs

Methods for handling the hybrid functionalities: methods for scanning the markers present on the board, handling of physical action following detection of played cards, integration with the board digital representation.

Assets/_Scripts/ServerJob.cs

This is an asynchronous service class that interacts with the remote server and services, in particular gets the validation XML answer for any card configuration.

Assets/__Scripts/AlertManager.cs

Methods for displaying and managing the alert popup and retrieving message texts from the KB.

Implementation Examples

Add a new scene between the “LoadGames” and the “Game” scene

Once you created a new scene, you can jump to it in code using the **SceneManager.LoadScene(string sceneName)**

native method or the

SceneChanger.ChangeScene(string sceneName)

implemented method.

If you need to introduce an Introductory scene between the “LoadGames” and the “Game” scene just if the chosen game level is 1, you can do that once all the game setup has been executed. Currently the scene is changed in the LoadGamesManager.NewGame() method after cards have been set-upped, in a callback:

```
FourTManager.I().CardsDoSetup(()=> {
            SceneManager.LoadScene("Game");
        }, FourTManager.I ().Game.SheetId);
```

You can change the flow like this:

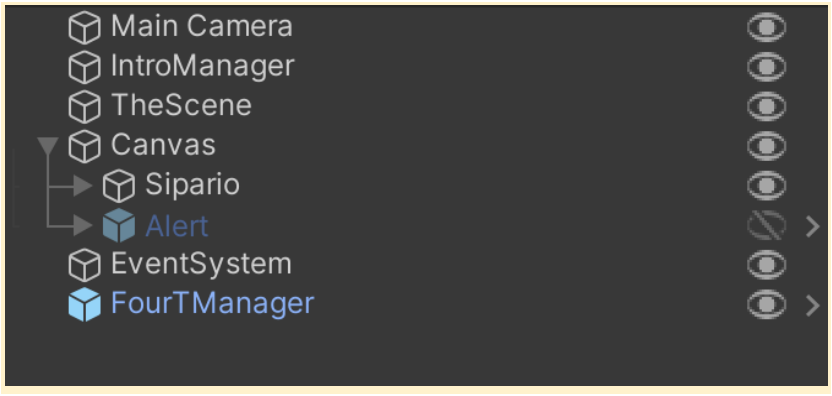
```
FourTManager.I().CardsDoSetup(()=> {
        if (FourTManager.I().Game.Level == 1)
            SceneManager.LoadScene("HelpIntro");
        else
            SceneManager.LoadScene("Game");
```

```
        }, FourTManager.I ().Game.SheetId);
```

Where **HelpIntro** is the name of the new scene you introduced.

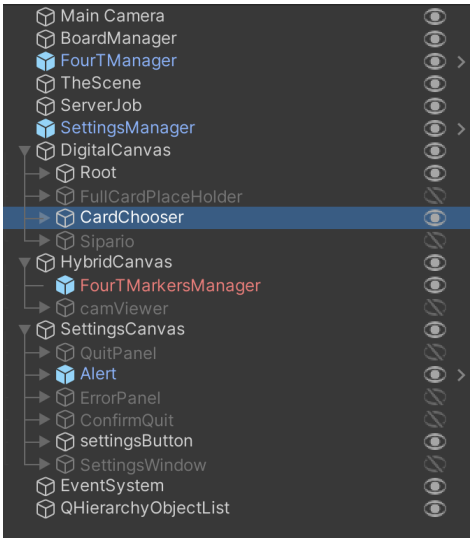
In the new scene you then should let the user to continue by loading the “Game” scene, either with a button action or with an event, by calling **SceneManager.LoadScene("Game");**

Any new scene must contains the **FourTManager** and the **Alert** prefabs and should be structured as displayed in the following image:



As best practice, the new scene should have a corresponding Class to manage the UI and any scene functionality calling it **[myNewSceneName]Manager** so if your new scene is called HelpIntro your Class will be named **HelpIntroManager**.

Change the choose cards flow



At the time the flow displays the card chooser popup filled with the available cards of the correct type. This is managed by the **BoardManager.I.ShowCardChooser()** method. This method gets all the available cards by the selected type (**FourTManager.I().Game.CardsAvailable(Type)**) and fills the slider with those cards if available.

If you need to change this flow introducing a new step before the slider you should use the Card Dictionary generated by the public **Dictionary<int, int> CardsAvailable(Card.CardType type)** method for the cards list; create a new popup element where you will print the cards and make all the methods to collect the cards to display in the slider. You then must change the code to display the chosen cards instead of all of them passing the new Dictionary as **cardsToDisplay** in the cards loop at **line 656**. Of course you should create a new method where to move all the code that is now used to open the slider and display the cards and invoke it once the player chooses the cards.

Creating a new popup type

A good example to implement a new popup can be the **AlertManager** Class that is used to display the most of the warnings of the application.

As for the AlertManager a Prefab element is needed for the popup UI.

Your new Class should have an Opener and a Closer method (see the **AlertManager** Class) to be invoked when needed.

Adding score to the gameplay

Create a new Class to keep track of the score as for example a **ScoreManager**.

Implement methods to add or subtract points from the Score.

If the score depends on errors or good choices you can modify the score when the Consistency Check or the Completeness Check is performed in the **BoardManager Class**.

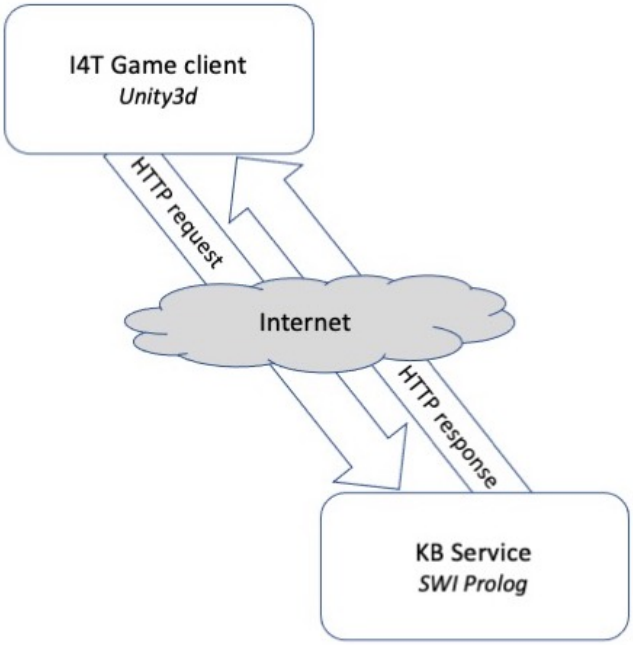
The Knowledge Base

The Knowledge Base (KB from here on) service is an HTTP based service implemented in Prolog and run through the SWI Prolog environment.

KB is accessed by the I4Ts game client through HTTP requests. KB responds by providing validation results for the current game board, as well as suggestions when requested by the user through the use of special cards.

Notice that the I4Ts game client and KB services don’t need to be co-located; the KB service can be run on nodes which are remote to the I4Ts game client, as long as the client is able to submit HTTP requests to KB.

The following figure provides a high level description of the communication flow between the I4Ts game client and KB.



A typical request to KB contains:

- the game level currently in use (entry or advanced)
- the current placement of the cards on the board
- whether KB should check the board for completeness or not.

The request details are specified as parameters of the HTTP request (the verb of the request - GET/POST or else, it doesn’t matter).

KB responds with an XML structure which contains the validation results and, if requested, suggestion cards.

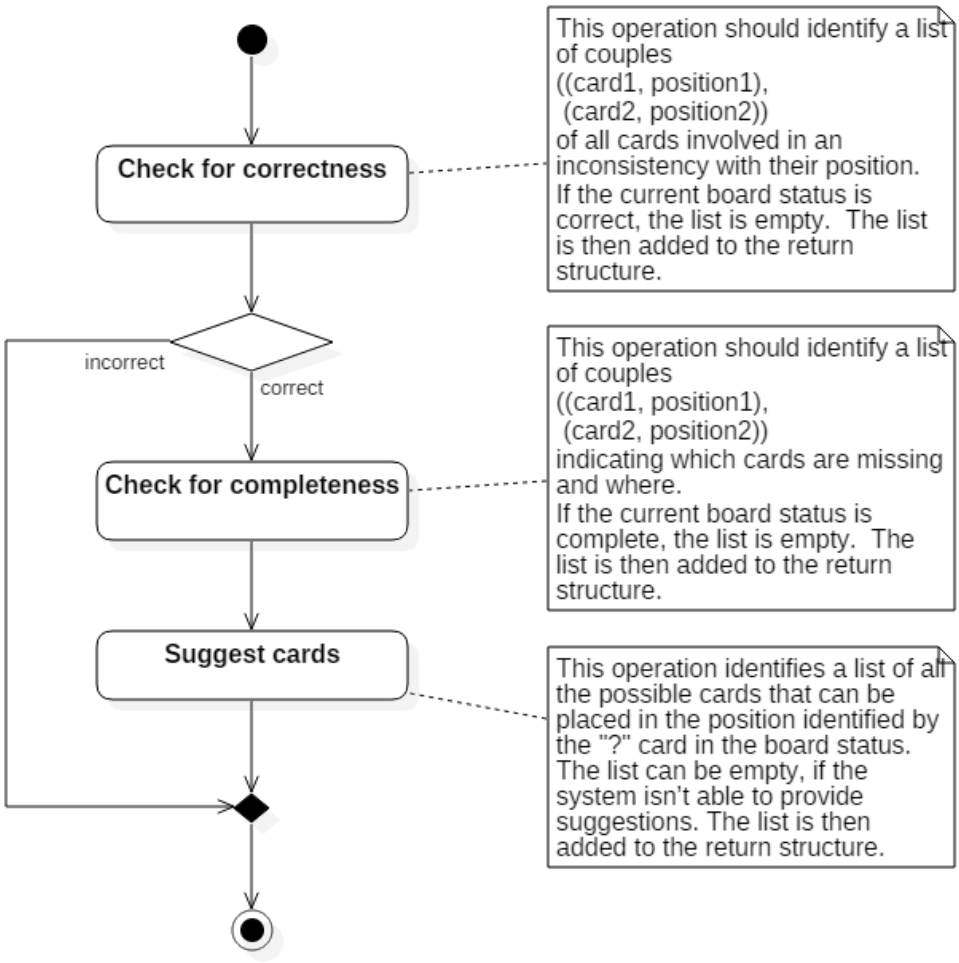
Internally KB’s behaviour changes significantly depending on the game level currently in use.

For “entry” (L1) level, KB’s main task is to match the provided board layout against one of the board templates which have been compiled as “valid” (i.e. the collaborative techniques); that’s where the use of a language like Prolog comes in very handy, as Prolog is particularly well suited for performing pattern matching of that kind.

For “advanced” (L2) level, KB performs a number of atomic operations - like performing a finer grain pattern match against known good patterns to verify that specific individual combinations of tasks/teams and technologies are valid, but also a number of lighter weight checks which grant more flexibility to the overall patterns that a user may be specifying.

In both cases, KB provides as many details as possible about detected errors and communicates them back to the caller through error code numbers and a default description.

The following figure provides an example of how KB “reasons” through a client request.



KB technical details

As most of the tasks performed by KB are based on pattern matching, KB is entirely implemented in Prolog, and exposed through a simple HTTP interface.

KB requires SWI-Prolog (https://www.swi-prolog.org/), which is available for free and supports the ability of exposing Prolog rules through a simple HTTP server out of the box.

KB’s internal components are roughly split into three modules:

- The HTTP interface
- The “base level” board analyzer
- The “advanced level” board analyzer.

The following sections get into more details for each of them, focusing mostly on how a developer can modify the existing KB behavior to better suit their needs.

HTTP Interface

KB relies on the SWI-Prolog HTTP server libraries (https://www.swi-prolog.org/pldoc/man?section=httpserver).

By executing the “swipl loadAll.pl” command, the server is automatically started on port 8000; you can verify that the server is listening for request using the “ping” endpoint:

```
$ curl -X GET localhost:8000/ping
<?xml version="1.0"?>
<!DOCTYPE ping-response >
<ping-response>
  <system-is-active />
  <system-version>4.42</system-version>
  <system-date>Mar 05, 2021</system-date>
  <client-ip>ip(127,0,0,1)</client-ip>
</ping-response>
```

The endpoint used to verify the status of a board is “semantic_check_Unity”; http_driver:buildBoard/2 provides the complete description of the HTTP parameters expected by the “semantic_check_Unity” endpoint, like the game “level” (base or advanced), whether the board should be checked for completeness or just for errors, and the details of the cards currently on the board in each the available slots of each of the four columns.

This simple example relies on default for all fields (“base level”, no completeness check, empty board slots), except for the presence of two cards in the first column, where the “task” card is wrong:

```
$ curl -X GET "localhost:8000/semantic_check_Unity?C1-CSW-TECHNIQUE=181&C1-ABA-CSS-TASK=138"
<?xml version="1.0"?>
```

```
<!DOCTYPE kb-response SYSTEM "kb_out.dtd">
<kb-response>
  <inconsistent-slots>
    <slot>C1-ABA-CSS-TASK</slot>
  </inconsistent-slots>
  <inconsistent-details>
    <code>202</code>
      <detail>This Task is not the one expected in this slot by this instance of Technique. Read the Technique card again and make a different choice.</detail>
  </inconsistent-details>
  <missing-cards>
  </missing-cards>
  <suggested-cards>
  </suggested-cards>
</kb-response>
```

“Base Level” Board Analyzer

“Base Level” and “Advanced Level” analyzer share content in the “kbCommons” module. There you will find most of the mappings among “Ts” and card IDs (card_instances_commons/2), the list of tasks (task/2), teams (team/2) and technologies (technology/2), as well as the list of error codes with their default description in English and some common utilities.

The core of the “Base Level” board analyzer is defined in the “kbBase” module, including the definition of the technique cards (card_instances_base/2), the related techniques (technique/2) and the main analyzer rule (checkBoard_base/3).

The analyzer relies on matching the provided board state against a set of valid patterns that are described by the boardPattern/1 facts. “boardPattern/1” are responsible for identifying *all* valid patterns in terms of the technique, task, team and technology IDs which are admissible in each column’s slot. For example, this is a valid pattern in the context of the “peer review” technique:

```
boardPattern(
  [
    column(1, '003', ← technique, week 1
              '102', '201', '306', '', ← task, team, technology
              '', '', '', ''),
    column(2, '', ← week2 of the same technique
              '112', '202', '311', '310',
              '', '', '', ''),
    column(3, '004',
                                 '105',  '202',  or('301','308'),
or('','301','308'),
                                 ↗ valid technology combinations
              '', '', '', ''),
    column(4, '005',
              '112', '202', '311', '310',
              '108', '206', '309', '310')
  ]).
```

The “boardPattern/1” facts are “expanded” when the system starts up into all the individual combinations allowed by the pattern definition; that happens automatically as part of loading the kbBase module, when the “assertAllPatterns/0” is executed. All expanded patterns are serialized into the “boardPatternExpanded.pl” file.

For example, in the scenario listed above, expanded patterns will be created to take into consideration the fact that the “306” technology in week 1 can be defined either in the first or the second technology slot, as well as to create the atomic entries for each of the acceptable combinations defined by the “or()” conditions on technologies of week 3.

If you need to add/delete/change valid patterns, or deal with the creation of new “T” instances, you will need to add/delete/change the corresponding patterns in kbBase.

Reloading KB always triggers the expansion of all defined patterns.

“Advanced Level” Board Analyzer

Similarly to the “base level” board analyzer with kbBase, the “advanced level” analyzer relies on rules defined in the kbCommons module and it exposes its core functionality in the kbAdvanced module.

The main board validation entry point for the “advanced level” analyzer is “checkBoard_advanced/5”.

The KB “advanced level” leaves more freedom to board users; as such, the KB checks are not strictly based on pattern matching against a set of predefined valid boards, but they rely on a series of verifications to validate that users are following specific rules and restrictions while defining their boards.

The analyzer performs four high level board checks:

1. It verifies that the task/team/technology and time combinations are acceptable.
   For example, the “assuming roles” task can only be performed by a “small group” team, and it takes a whole week if the “forum” technology is used, but it can be formed along with another task in the same week if “videoconference” or “no technology” are used.
   Such valid combinations are captured in the kbAdvanced:ttt/4 facts; for the example above:
   ```
   % Assuming roles
   ttt('111', '203', '301', 1).   % forum
   ttt('111', '203', '305', 0).   % videoconference
   ttt('111', '203', '310', 0).   % no tech
   ```
   Similarly to what happens to “base level” patterns, these patterns can contain combinations of valid “T’s”, and are expanded automatically when KB is loaded; this is an example of a pattern relying on expansion:
   ```
   % Commenting on someone else's work
   % text editor or wiki
   ttt('105', or(['201', '202', '203']), or(['308','304']), 1).
   % (text editor or wiki) and forum
   ttt('105', or(['201', '202', '203']), and(['301', or(['308', '304'])]), 1).
   % (text editor or wiki) and no tech
   ttt('105', or(['201', '202', '203']), and(['310', or(['308', '304'])]), 1).
   % forum
   ttt('105', or(['201', '202', '203']), '301', 1).
   ```
   These patterns can be modified, and new patterns added, as needed;

their expansion is performed when KB is loaded by the “kbAdvanced:expand_all_TTTs/0” rule.

2. It verifies that the combinations of teams and technologies used are acceptable.
   That’s controlled by the kbAdvanced:valid_techs_for_team/2 rules, where acceptable technologies for each team are enumerated.
3. It verifies the compatibility of technologies used in the same context.
   That’s controlled by the kbAdvanced:incompatible_techs/2 rules, where each technology lists the technologies it cannot be used side by side.
4. It checks that prerequisite tasks are included before the specified task.
   That’s controlled by the kbAdvanced:prerequisite_tasks/2 rules, where each task is associated with a list of other tasks where at least one of those must be specified on the board before the task at hand.

These individual checks allow KB to provide detailed error information when users specify invalid boards.