Fundamentals of Artificial Intelligence exam project
academic year 2021/2022

ArtHelper

A tool for helping users in studying artistic items

Teacher: Professor Stefano Ferilli
Student: Francesco Benedetti

Table of Contents

1: Introduction

The artistic field interests many people, for study, tourism, curiosity or passion. However, understanding an artwork isn't always straightforward, because there are many notions that the user is required to know in order to best understand it. Take for example the "School of Athens" fresco by Raphael. Looking at the opera is not enough to grasp all the background information needed to complettely understand it. This information concern its author, the style that the fresco follows, who are the people portrayed, where it's placed and if there are other paintings that can help to understand the motivations behind it.

Finding out which these notions are can be time consuming due to the vastity of the artistic field. ArtHelper's goal is to help the user in this phase, by listing what he needs to know in order to completely understand an item. In this way, the user won't have to blindly search for things that may even turn out to be not relevant for the item he wants to seek information about.

1.1: Overview of the system

The user can ask for information about items belonging to these four classes:

- Artworks
- Artists
- Churches
- Styles

The user can search for an element belonging to any of these classes, and the system will answer by listing the general information about it and items to which the element is related according to rules that will be later discussed. In some cases the user can also ask why the system is suggesting an item, and he will get an explanation.

2: Pre-requisites

This section gives an overview about the pre-requisites that were detected for each class of items. This will help to better understand the conceptualization choiches that were made.

2.1: Artworks

The information pre-required to understand an artwork is:

- Title;
- Author;
- Artistic style(s) it follows;
- City and year in which it was made;
- Museum that owns it;
- A short description;
- Main characters portrayed in the artwork, if any;
- Secondary characters portrayed in the artwork, if any;
- Related artworks. Two artworks are related if:
    - They are part of the same polyptych
    - They potray the same characters and follow the same style (i.e: “Venere” and “Primavera”, both by Botticelli)
    - They are two frescoes made by the same author and are in the same place (i.e: “School of Athens” and “Disputa del sacramento”, both made by Raphael in the Vatican rooms).

2.2: Artists

When talking about artists we refer to both artists and architects. The following notions are necessary to properly understand an artist:

- Name;
- Year of birth and year of death;
- Artists that influenced him/her;
- The artworks he made and/or the churches he designed;
- List of styles he followed.

2.3: Churches

Churches are a subclass of a broader class named Museums. This was done because churches very often own artworks, which makes them a kind of museums. Classic museums like the Van Gogh museums are interesting for the operas they own, but their structure is not an attraction since they are simple buildings. Churches, on the other hand, represent attractions themselves. This is why it was decided to focus more on churches. The following information is required to understand a church:

- Name;
- City where it is located;
- Construction starting year and ending year;
- Architect(s) who designed the church, if known;
- Style(s) that it follows;
- Artworks owned by the church, if any;
- Related churches. Two churches are related if:
  - They follow the same style and are located in the same city, since they are two representations of the artistic school of thought that a city had about a style;
  - They were built in the same city during the same years. It may be possible that the way in which a church was built influenced the way in which the other was built.

2.4: Styles

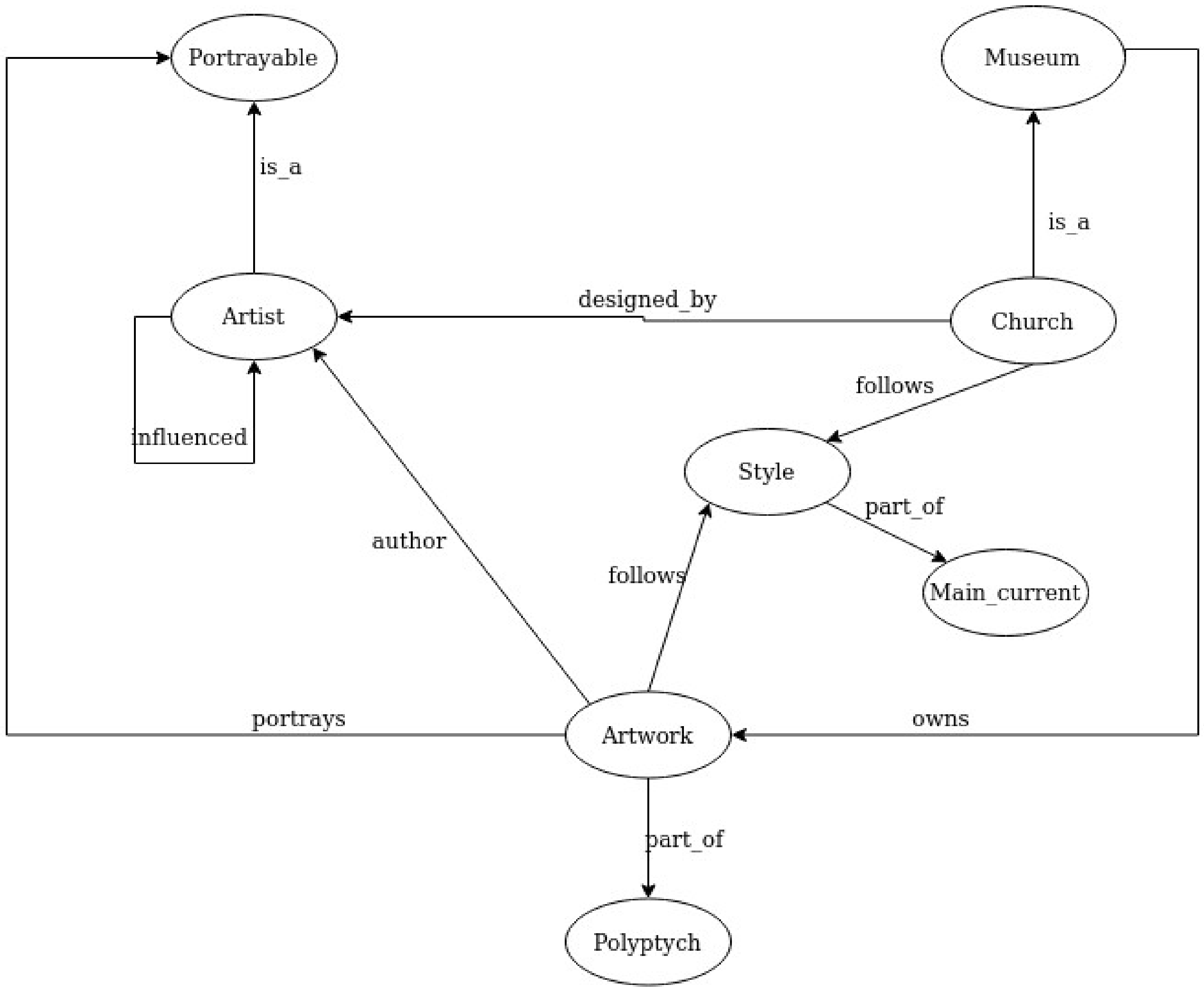
We decided that this information is needed to properly understand a style:

- Style name;
- Year when it started, year when it ended;
- Artists exponent of that style;
- Other styles that influenced it. A style S1 influenced the style S2 if:
  - They co-existed, which means that they may have influenced each other;
  - They belong to the same main current and S1 is older than S2 (ie proto Renaissance, early Renaissance and high Renaissance all belonged to the Renaissance current. Proto Renaissance influenced the other two);

- There is a monument or an artwork that follows both the two styles, the oldest influenced the youngest.

3: Conceptualization

In this section we will analyze and explain in detail how the knowledge about the domain is organized. The following schema depicts the entities of interest as well as the relationships between them:



Comments:

When discussing the rules, the **portrayable** entity was not mentioned. It represents any character that can be portrayed in an artwork. A portrayable can be a real person as well as somebody belonging to mythology or religion. Since artists can also be portrayed in operas, the choice of making the Artist entity a subclass of the Portrayable entity was made.

The **portrays** relationship between Artwork and Portrayable is actually made of two relationships: **main_character** and **secondary_character**. We will use the two to link an artwork to the characters depicted. The former will be used when the character plays a central role in the artwork, the latter will link the artwork to characters that are not so important. This information could be useful to the final user to detect which are the characters to study more accurately. For instance, in the painting “Nascita di Venere” by Botticelli, the main character is the Venere goddess while the secondary characters are Zefiro and the Clori.

Here is a list of all the relationships with the respective cardinality:

- An artist can be influenced by zero, one or more artists.
- A church can be designed by zero, one or more artists. The zero case happens when the architects are not known.
- A church must follow at least one style.
- An artwork must follow at least one style.
- A style can be part of at most one main current.
- The author of an artwork is exactly one artist.
- An artwork can portray zero, one or more portrayables. If a portrayable is the main_character of an artwork, he can’t also be a secondary_character for the same artwork, and viceversa.
- An artwork can be part of at most one polyptych.
- A museum can own zero, one or more artworks.

3.1: Entity features

The following table lists, for every entity, its features.

Entity name	Features	Comments
museum	museum name, city where it’s placed	

church	church name, city where it’s placed, starting construction year, ending construction year	
portrayable	name	
artist	name, birth year, death year	
style	name, year when it began, year when it ended, field	The “field” feature is used to detect potentially related styles. It can have three values: “architecture”, “art” or “both”, depending on the style being principally architectural, artistic or both. Two styles cannot be related if one is exclusively artistic and the other is exclusively architectural.
main_current	list of the styles it involves	
artwork	title, year, city where it was done, type, short description	The “type” field describes the type of artwork. Currently there are only two types of artwork: painting or a fresco.
polyptych	list of the artworks it comprehends.	

3.2: Facts

The knowledge about entities is formalized by the following facts:

- **museum(M, N, C)**: The museum M has name N and is located in the city C.
- **church(C, N, Ci, Ys, Ye)**: The church C has name N and is located in the city Ci. Its construction began in year Ys and ended in year Ye.
- **portrayable(P, N)**: The portrayable P has name N.
- **artist(A, N, Yb, Yd):** The artist A has name N, was born in the year Yb and died in the year Yd.

- **style(S, N, Ys, Ye, F)**: The style S has name N. It started in the year Ys, ended in the year Ye and concerned the field F.
- **main_current([S1, …, Sn])**: The styles S1, …, Sn belonged to the same main stylistic current.
- **artwork(A, T, Y, C, D)**: The artwork A has title T, it was made in the year Y in the city C. It has description D.
- **polyptych([A1, …, An])**: The artworks A1, …, An belong to the same polyptych.

3.3: Predicates

The following predicates state relationships between entities:

- **author(AW, A)**: The artist A is author of the artwork AW.
- **designed_by(C, [A1, …, An]):** The church C was designed by the artists A1, …, An
- **owns(M, [AW1, …, Awn]):** The museum M owns the artworks AW1, …, AWn
- **main_subject(AW, [P1, …, Pn]):** The portrayables P1, …, Pn are the main subjects of the artwork A.
- **secondary_subject(AW, [P1, …, Pn])**: The portrayables P1, …, Pn are the secondary subjects of the artwork A.
- **follows(I, [S1, …, Sn]):** The item (artwork or church) I follows the styles S1, ..., Sn
- **influenced_by(A, [A1, …, An]):** The artist A was influenced by the artists A1, …, An

3.4: Rules

In the following, we list the rules that the system will use to answer a query.

3.4.1: Rules concerning the styles

- **is_exponent(+A, +S)**: Returns true if the artist A is exponent of the style S. It checks that at least one of the artworks made by A follows S.

- **is_exponent(+Architect, +S)**: Works the same way as the one above, but instead of checking artworks it checks at the churches that an artist designed. The construction of a church can span through centuries and follow several styles, so we have to make checks on the years of birth/death of the artist and years of begin/end of a style, to avoid having an artist follow a style that started after his death. For instance, the “Santa Maria del Fiore” church took more than 600 years to complete and follows Gothic and Neogothic styles. Without the checks, we would have that Filippo Brunelleschi, its first architect, who died in 1446, followed Neogothic, which would be impossible since the style started in 1790 and was followed by Emilio De Fabris, the last main architect of the church.
- **potentially_related_fields(+F1, +F2):** Utility rule used to check whether two fields are related. Remind that a style must have one of the following fields: *“artistic”, “architectural”, “both”*. Two fields are related if they are the same or if at least one of them is equal to *“both”*.
- **co_existing_styles(+S1, +S2):** Returns true if S1 and S2 co-existed for some time and if their fields are potentially related. It is very likely that styles which co-existed had some sort of influence on each other, for example in the past a style could start as “answer” to another one, hence the latter becomes a prerequisite needed to fully understand the former.
- **influenced_same_current(+S1, +S2)**: true if S1 started after S2 and if they both belong to the same main current.
- **influenced_same_art(+S1, +S2, -A):** true if the style S1 started after the style S2 and if there is one artwork or a place A that follows both the styles. For explainability reasons, A is also returned.

3.4.2: Rules for churches

- **churches_same_style_and_city(+C1, +C2, -City, -Styles):** True if the churches C1 and C2 are located in the same city and follow some styles in common. For explainability reasons, the City and the Styles are also returned.
- **churches_same_construction_years(+C1, +C2, City):** true if the churches C1 and C2 are located in the same city (which is also returned) and were built during the same years.

- **retrieve_church_information(+C, -N, -City, -Yb. -Ye, -Architects, -Styles):** Given the church C, returns its name N, the City where it's located, the year Yb when its construction began, the year Ye when the construction ended, the list of architects and the list of styles it follows.

3.4.3: Rules for artworks

- **other_elements_polyptych(+AW, -Artworks):** If the artwork AW belongs to a polyptych, returns the other artworks that belong to the same polyptych.
- **artwork_same_subject(+AW1, +AW2, -Characters, -S):** Given the artworks AW1 and AW2, returns true if the artworks follow the same Style (which is returned) and if at least one of the main subjects of AW1 are also portrayed (as main or secondary subjects) in AW2, since it means that the artworks may be related. The characters in common are returned.
- **fresco_same_place(+AW1, +AW2, -Museum, -Artist):** True if the artworks AW1 and AW2 are of type "fresco", are placed in the same Museum and were made by the same Artist. For explainability reasons the museum and the artist are also returned.
- **retrieve_information_artwork(+AW, -T, -Y, -C, -Type, -S, -A, -MainSubjects, -SecondarySubjects, -M, -D):** Given the artwork AW, returns its title T, the year Y when it was made, the city C where it was created, its Type, the styles S that it follows, the author A, the list of main subjects, the list of secondary subjects, the museum M that owns it and its description D.

3.4.4: Rules for artists

- **retrieve_artist_places(+A, -P):** Given an artist A, returns the places P that he designed.

4: Inference

The knowledge about the domain and the rules was coded using Prolog. An ad-hoc backward chaining strategy was implemented in order to make the system able to provide explanations for the conclusions it makes. The general idea behind the explanation mechanism is to store, during the inference, all the rules and facts that are successfully used, and show them when asked to provide an explanation. A proper formalism for distinguishing between rules and facts is required. In this formalism, facts are stated according to the following schema:

```
fact(entity_name, (id, Arguments))
```

For instance, this is how we declare an Artist entity (*vincent_van_gogh* is the ID):

```
fact(artist, (vincent_van_gogh, ‘Vincent Willem Van Gogh’,
1853, 1890)).
```

While this is how we declare an Artwork entity:

```
fact(artwork, (sunflowers, ‘Sunflowers’, 1888, ‘Arles’,
painting, ‘…’)).
```

To say that Van Gogh is the author of the “Sunflowers” painting:

```
fact(author, (sunflowers, vincent_van_gogh)).
```

Rules:

The syntax for declaring a rule is:

```
rule(rule_name, (Arguments), [Rule body])).
```

For instance, the rule to determine whether an artist is exponent of a style is written as follows:

```
rule(is_exponent,
   (Artist, S),
   [
      fact(artist, (Artist, _, _, _)),
      fact(author, (Work, Artist)),
      fact(follows, (Work, LS)),
      call(!),
      fact(style, (S, _, _, _, _)),
      call(member, (S, LS))
     ]).
```

The *call(operation)* uses the built-in *call* predicate of Prolog, which queries the predicate expression given as argument.

4.1: Backward chaining implementation

The basic operation of our backward chaining engine is named *backward.* This is the operation that the final user can use to start an inference process*:*

```
backward(fact(FactName, Arguments)) :-
   !,
   fact(FactName, Arguments).
backward(rule(RuleName, Arguments)) :-
   !,
   rule(RuleName, Arguments, Body),
   bw(rule(RuleName, Arguments, Body), [], L),
   asserta(used(L)).
```

The backward called on a fact simply verifies that the fact is in the knowledge base. When we call it on a rule, the situation is different. We first need to retrieve the rule body from the KB and then call the *bw* procedure to check if all the conditions are satisfied. It takes as input a list, which at the beginning is empty. This list will contain all the rules and facts used to satisfy the original rule. If it succeeds, the list L is asserted in the knowledge base, so the user can consult it.

The *bw* implementation is the following:

```
bw(fact(Z, P), L, L1) :-
    !,
    backward(fact(Z, P)),
    append([(Z, P)], L, L1).
/*Backward of two conditions in or. FR means that the
argument can be either a fact or a rule */
bw((FR1; FR2), L, L1) :-
    !,
    (bw(FR1, L, L1);
    bw(FR2, L, L1)).
/*Backward of a call */
bw(call(F), L, L) :-
    !,
    call(F).
/* The following two rule are used to prove a rule. The first
one retrieves the rule body and calls the second to prove
it*/
bw(rule(RuleName, Arguments), L1, L2) :-
    !,
    rule(RuleName, Arguments, RuleBody),
    bw(rule(RuleName, Arguments, RuleBody), L1, L2).
bw(rule(RuleName, Arguments, RuleBody), L, L2) :-
    !,
    prove(RuleBody, L, L1),
    append([(RuleName, Arguments)], L1, L2).
```

This is how the *prove* procedure is implemented. It checks every element in the rule body and backwards on it.

```
prove([], L, L) :- !, true.
prove([true], L, L) :- !, true.
prove([false], L, L) :- !, false.
prove([H|T], L, L3) :-
    !,
    bw(H, L, L2),
    prove(T, L2, L3).
```

In order for the inference process to start, the queries must be submitted in the following syntax: **backward(rule, rule_name, params).** After getting the answers, the explanation can be consulted with the query **used(X),** which will return the list of facts and rules used. After reading the explanation, it is suggested to retract it because if we make an other query we may make confusion and mistakenly read the wrong explanation. The explanations are retracted with the command **retractall(used(X)).** which uses the Prolog built-int predicate **retractall.**

An important observation must be done before proceeding. When talking about the rules, we said that some of them contain additional arguments, used "for explainability reasons”. If we used the system from the Prolog command line, these variable wouldn’t be needed. Take, for example, the rule **influenced_same_art(S, S1, Artwork).** The Artwork variable wouldn’t be needed if we directly used the system from Prolog, because we could deduce its value by reading the explanations.

Query:

```
backward(rule(influenced_same_art, (early_renaissance, S))).
```

Answer:

```
S=gothic.
```

Explanation:

```
used(X).
X = [(influenced_same_art, early_renaissance, gothic),
(follows, santa_maria_novella, [early_renaissance, gothic]),
(style, gothic, 'Gothic', 1144, 1600, architecture),
(style, early_renaissance, 'Early Renaissance', 1400, 1490,
both)].
```

By reading the explanation, we can easily see that the system answered “Gothic” because there is the Santa Maria Novella church that follows both Gothic and early Renaissance, making the two styles related.

However, the system is endowed with an user interface created with Python libraries (discussed in section 6). The library **pyswip**, used to query a Prolog KB using Python, is structured in a way that makes very hard reading the list of explanations because it is returned as string. This is why, for semplicity reasons, it was deemed as convenient putting additional variables to the rules that simplify the process of providing the user an explanation, because in this way everything that is needed to produce an explanation will be found in the first element of the list.

5: Knowledge base

In the knowledge base are represented:

- 12 styles;
- 32 artworks;

- 16 museums, 7 of which are churches;
- 17 artists and 46 more portrayables.

The items mainly date back to the Medieval period, but a part of them is more recent (until the post-impressionism era). The KB can be easily enriched with new facts and rules as long as they are represented following the correct syntax.

The modules can be found at the path **ArtHelper/prolog/kb**.

5.1: Modules

styles.pl

Module containing the definitions of styles and main currents, as well as predicates stating the styles followed by every artwork and church.

```
%Definition of a style
fact(style, (gothic, 'Gothic', 1144, 1600, architecture)).
%Definition of a main current
fact(main_current, ([gothic, neogothic])).
%Stating which styles the church follows
fact(follows, (santa_maria_del_fiore, [gothic, neogothic])).
```

people.pl

Contains information about artists and portrayables, and the predicates stating the artists that influenced an other artist.

```
%Definition of an artist
fact(artist, (raffaello, 'Raffaello Sanzio', 1483, 1520)).
%Raffaello was influenced by  Paolo Uccello and Michelangelo
fact(influenced_by, (raffaello, [paolo_uccello,
michelangelo])).
%Definition of a portrayable
fact(portrayable, (federico_montefeltro, 'Federico da
Montefeltro')).
```

museums.pl

The definition of museums and churches are contained in this module. It also contains predicates for the *owns* and *designed_by* relationships.

```
%Definition of a museum
fact(museum, (louvre, 'Louvre', 'Paris')).
%Definition of a church
fact(church, (santa_croce, 'Basilica di Santa Croce',
'Florence', 1294, 1385)).
%The Louvre owns the following operas contained in the KB
fact(owns, (louvre, [monna_lisa, san_romano_louvre,
madonna_piot])).
%Designer of the Santa Croce church
fact(designed_by, (santa_croce, [arnolfo_di_cambio])).
```

art.pl

This file contains the definition of artworks, polyptychs, the author of each artworks and the subjects portrayed in every artwork.

```
%Artwork definition
fact(artwork, (primavera, 'La Primavera', 1480, 'Florence',
painting, '..')).
%A polyptych
fact(polyptych,
([battista_sforza_portrait,federico_montefeltro_portrait])).
%Author of an artwork
fact(author, (primavera, sandro_botticelli)).
%Main subjects of the 'Primavera'
fact(main_subject, (primavera, [venere])).
%Secondary subjects of the 'Primavera'
fact(secondary_subject, (primavera, [flora, zefiro, clori,
mercurio, cupid, grazie])).
```

If an artwork doesn't portray a main subject and/or a secondary subject, we have to explicitly state that by adding the predicates main_subject and secondary_subject concerning that artwork and putting an empty list as second argument. For example, the 'Sunflowers' painting by Van Gogh does not portray any subject.

```
fact(main_subject, (sunflowers, [])).
fact(secondary_subject, (sunflowers, [])).
```

utils.pl

Contains three utility functions:

- **list_concat(L1, L2, L3):** Takes the lists L1 and L2 and concatenates them to obtain L3;
- **list_intersect(L1, L2, I):** Returns the intersection I between the list L1 and L2;
- **convert_list_elements_to_names(L, Lnames):** The relationships between items are stated by adding predicates with their ids. This choice was made because working directly with the item names could lead to confusion since in some cases the complete name of an artwork or an artist is very long, slowing down the coding and debugging process. Moreover, the name is not always a unique identifier since some artworks can have the same title. Retrieving the name of a single item is easy because we can just look at the fact containing all the information about it, but when we are dealing with lists of IDs the situation becomes more tricky. This utility rule takes a list of IDs and finds for each of them their name. We have six different implementations of this rule, one for each item type (artist, portrayable, artwork, church, museum, style).

rules_styles.pl

Contains the implementation of the rules described in the section 3.4.1.

- **rule(is_exponent, (Artist, S)):** This rule checks if the Artist followed the style S. It works by taking all the artworks made by the artist and seeing if at least one of them follows S. Alternatively, we can avoid binding S to a specific style. In this way, we will retrieve all the styles followed by an artist. The system will use the rule to do so.

  If we query:

  ```
  backward(rule(is_exponent, (raffaello, S))).
  ```

  We will get as result

  ```
  S=high_renaissance.
  ```

  since the KB contains only artworks made by Raphael which follow high renaissance.

- **rule(is_exponent, (Architect, S)):** As discussed before, this rule is similar to the previous one but makes some checks. Let’s look at the implementation:

```
rule(is_exponent, (Architect, S),
[
    fact(artist, (Architect, _, Yb, Yd)),
    call(!),
    fact(designed_by, (Place, Architects)),
    call(member(Architect, Architects)),
    fact(style, (S, _, Ys, Ye, _)),
    call(Yd >= Ys),   /*An architect cannot follow a
style that started after his death*/
    call(Ye >= Yb),   /*An architect cannot follow a
style that ended before his birth*/
    fact(follows, (Place, LS)),
    call(member(S, LS))
]).
```

- **rule(co_existing_styles, (S1, S2)):** Checks if the two styles co-existed for some time and if their fields could be related. By not binding the second argument to a specific style, we can use this rule to find all the styles that co-existed with the style S1.

```
rule(co_existing_styles, (S1, S2),
[
    fact(style, (S1, _, Yb1, Ye1, F1)),
    call(!),
    fact(style, (S2, _, Yb2, Ye2, F2)),
    call(S1 \= S2),
    rule(potentially_related_fields, (F1, F2)),
    /*Check the possible combinations that would make
the style co-existing*/
     (
        (
           call((Yb1 < Yb2, Ye1 > Yb2))
        );
        (
            call((Yb1 > Yb2, Ye2 > Yb1))
       )
     )
 ]
).
```

Example: this query will find all the styles that co-existed with mannerism:

```
backward(rule(co_existing_styles, (mannerism, S))).
```

And the answer is:

```
S = high_renaissance.
```

- **rule(influenced_same_current, (S1, S2)):** S2 is a pre-requisite for understanding S1 if they belonged to the same current and if S2 is older than S1. This additional check is needed because the older style is needed to understand the newer, while the opposite is not true. By not binding the second argument, we can use this rule to find all the styles that belonged to the same main current as S1.

  Example:

  ```
  backward(rule(influenced_same_current,
  (early_renaissance, S))).
  ```

  Answer:

  ```
  S = proto_renaissance.
  ```

- **rule(influenced_same_art, (S1, S2, I)):** S2 is a pre-requisite for understanding S1 if there is an item (artwork or church) I that follows both the styles and if S2 is older than S1. By not binding the second variable, we can use this rule to find all the styles that were used together with S1 for the making of any item.

  Example:

  ```
  backward(rule(influenced_same_art, (early_renaissance,
  S2, I))).
  ```

  Answer:

  ```
  S = gothic, I = santa_maria_novella.
  ```

  because the church Santa Maria Novella follows both Gothic and early Renaissance.

rules_churches.pl

This file contains the implementation of the rules in section 3.4.2

- **rule(churches_same_style_and_city, (C1, C2, City, Styles)):** By not binding the second variable we find all the churches in the same city as C1 and such that the intersection of the styles that C1 and C2 follow is not empty.

  Example:

  ```
  backward(rule(churches_same_style_and_city,
  (santo_spirito, C, City, S))).
  ```

  Answer:

  ```
  C=santa_maria_novella, City=”Florence”, S=[‘Early
  Renaissance’]
  ```

- **rule(churches_same_construction_years, (C1, C2, City)):** Find all the churches that were built in the same city where C1 was built and during the same years.

  Example:

  ```
  backward(rule(churches_same_construction_years,
  (san_frediano, C2, City)))
  ```

  Answer:

  ```
  C2=santa_maria_del_fiore, City='Florence';
  C2=santo_spirito, City='Florence.
  ```

- **rule(retrieve_church_information, (C, Name, City, Yb, Ye, A, S)):** Looks at the fact in the KB to retrieve the basic information about the church C.

rules_artworks.pl

This file contains the implementation of the rules discussed in section 3.4.3.

- **rule(other_elements_polyptych, (A, Artwork)):** Checks if the two artworks are different and are members of the same polyptych. By not binding the second variable, we will find all the artworks that belong to the same polyptych to which A belongs.

  Example:

  ```
  backward(rule(other_elements_polyptych,
  (san_romano_louvre, A))).
  ```

  Answer:

  ```
  A = san_romano_uffizi; A=san_romano_nationalgallery.
  ```

  (the three paintings belong to a triptych)

- **rule(artwork_same_subject, (AW1, AW2, Characters, S)):** The last two variables are never bound because they are used for explanation. If we avoid binding the second variable we will retrieve all the artworks that follow the same style as AW1 and portray at least one of its main characters.

  ```
  rule(artwork_same_subject, (AW1, AW2 Characters, S),
  [
     fact(artwork, (AW1, _, _, _, _, _)),
     fact(main_subject, (AW1, AW1main)),
     /*If the first artwork doesn't have main characters
  we can stop*/
     call(AW1Main \== []),
     fact(follows, (AW1, S)),
  ```

```
        call(!),
        fact(artwork, (AW2, _, _, _, _, _)),
        call(AW1 \= AW2),
        %Ensure that AW2 follows the same style as AW1
        fact(follows, (AW2, S)),
        fact(main_subject, (AW2, AW2Main)),
        fact(secondary_subject, (AW2, AW2Secondary)),
            /*We put all the characters portrayed by AW2 into
a unique list since we are not interested in distinguishing
between main and secondary characters for the second
artwork*/
        call(list_concat(AW2Main, AW2Secondary,
     AW2Portrays)),
        call(AW2Portrays \== []),
        call(list_intersect(AW1Main, AW2Portrays,
     Intersection)),
        call(Intersection \== []),
           /*Instead of returning the ids of the characters
in the intersection, we retrieve their names*/
         call(convert_list_elements_to_names(Intersection,
     Characters))
       ]
     ).
```

Example: we want to find the artworks that portray the same main subject as the “Nascita di Venere”:

backward(rule(artwork_same_subject, (venere_birth, A, C, S))).

Answer: There are two artworks that match the query: the first is the “Primavera” and the second is “Venere e Marte”

```
A=primavera, C=[‘Venere’], S=early_renaissance;
A=venere_marte, C=[‘Venere’], S=early_renaissance.
```

- **rule(fresco_same_place, (AW1, AW2, Place, Artist)):** If AW1 is a fresco, this rule is used to find all the frescos in the same museum as and made by the same author.

  Example:

  ```
  backward(rule(fresco_same_place, (athens_school, AW2, P,
  A))).
  ```

  Answer:

  ```
  AW2=sacrament_dispute, P=vatican_rooms, A=raffaello.
  ```

- **rule(retrieve_information_artwork, (AW, T, Y, C, Type, S, A, MainSubjects, SecondarySubjects, M, D)):** Returns the basic information about an artwork.

rules_artists.pl

This file contains the rule **rule(retrieve_artist_places, (A, P))** that returns the churches designed by an artist. We don’t need a similar rule to retrieve the artworks made by an artist because each artwork can be made only by an artist and there is a predicate for each artwork of which an artist is author.

Example:

```
backward(rule(retrieve_artist_places, (leon_battista_alberti,
P))).
```

Answer:

```
P=santa_maria_novella.
```

6: User interface

A graphic user interface was created for simplifying the interaction between the user and the system. The interface was entirely realized using the Python programming language and is made of two main blocks: the module that allows Python to interact with the Prolog knowledge base, and the actual interface.

6.1: Requirements

The following dependencies must be installed in order to correclty run the system:

- **Python 3.10**
- **SWI-Prolog**

The necessary Python libraries will be installed by running the command

```
pip install -r requirements.txt
```

The tkinter needs to be manually installed since it is not available on pip. On Ubuntu, it can be installed with the command

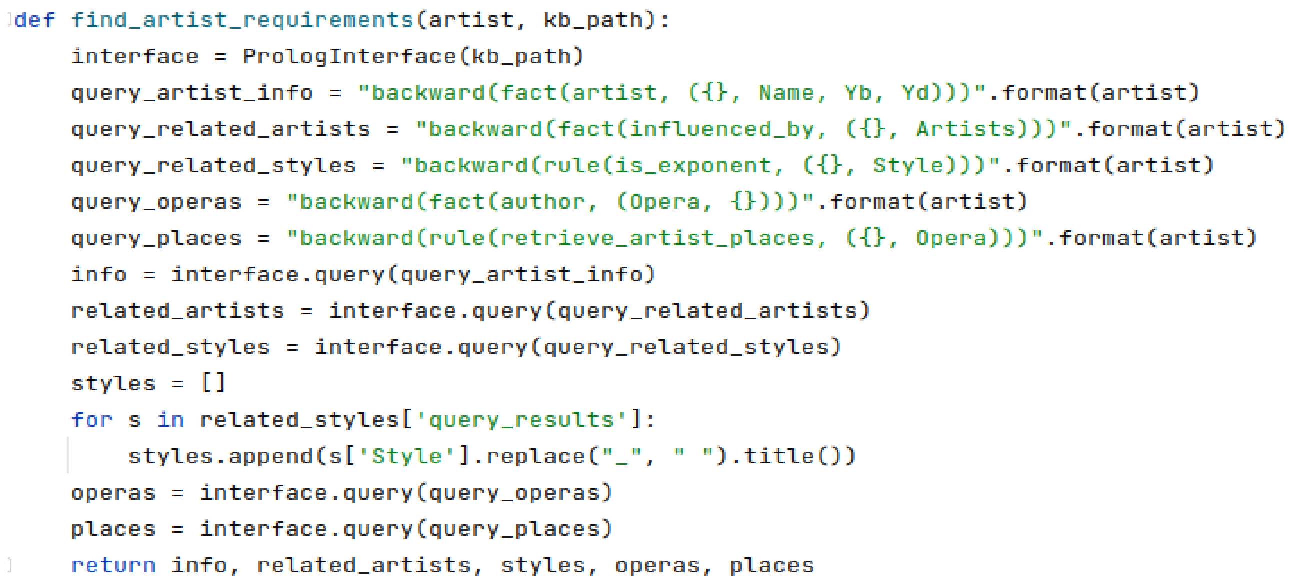
```
sudo apt-get install python3-tk.
```

6.2: Prolog interface

The interface between Python and Prolog is realized using the Pyswip library. When initialized, the interface first consults all the Prolog files. It also provides a function for querying the knowledge base. The output is a dictionary having one or two fields: the first contains the answer to the query, the second contains the explanation, if provided. The **query** function takes as input the query in string format. The module named **queries.py** contains the following functions (the last two are used for building the interface):

- **find_artwork_requirements(artwork, kb_path)**
- **find_artist_requirements(artist, kb_path)**
- **find_church_requirements(church, kb_path)**
- **find_style_requirements(style, kb_path)**
- **find_artworks_names(kb_path)**
- **find_names(kb_path)**

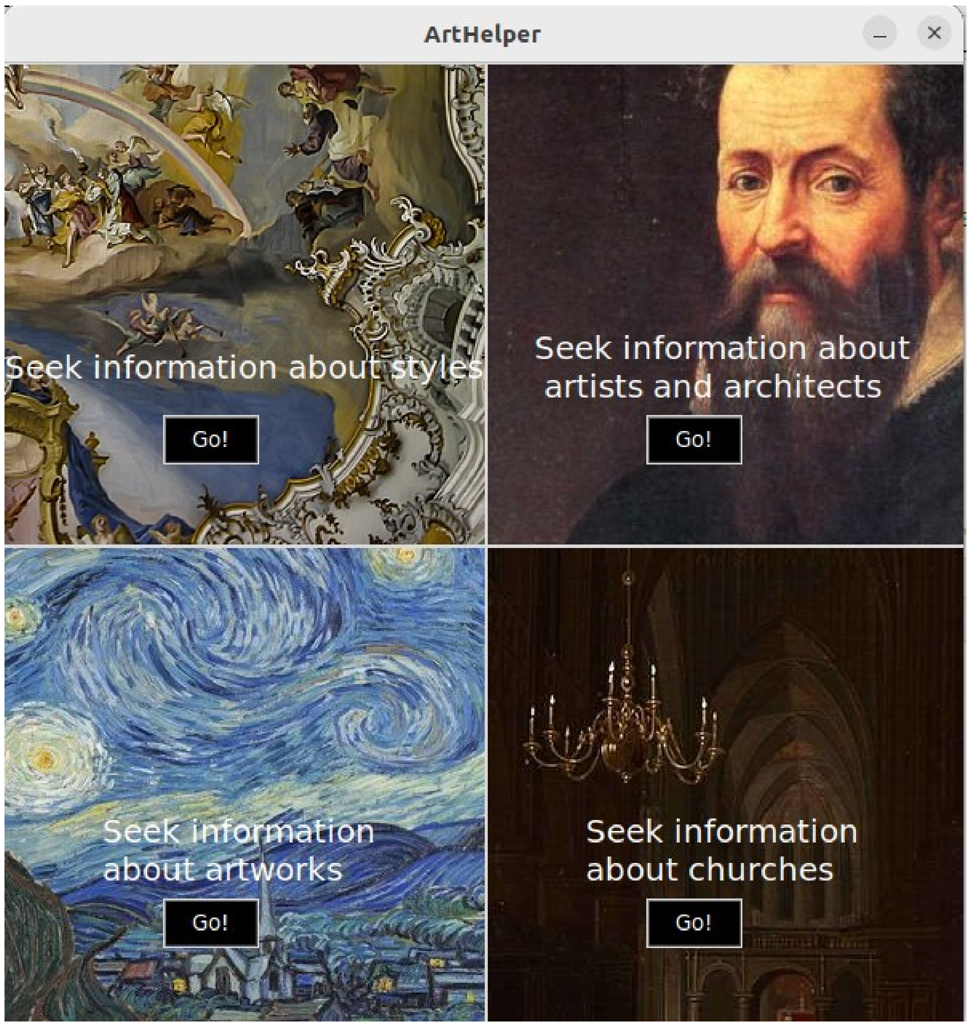
Each function executes all the queries needed to find the notions necessary to understand the item given as first argument. For example, this is the implementation of **find_artist_requirements.**



```
def find_artist_requirements(artist, kb_path):
    interface = PrologInterface(kb_path)
    query_artist_info = "backward(fact(artist, ({}, Name, Yb, Yd)))".format(artist)
    query_related_artists = "backward(fact(influenced_by, ({}, Artists)))".format(artist)
    query_related_styles = "backward(rule(is_exponent, ({}, Style)))".format(artist)
    query_operas = "backward(fact(author, (Opera, {})))".format(artist)
    query_places = "backward(rule(retrieve_artist_places, ({}, Opera)))".format(artist)
    info = interface.query(query_artist_info)
    related_artists = interface.query(query_related_artists)
    related_styles = interface.query(query_related_styles)
    styles = []
    for s in related_styles['query_results']:
        styles.append(s['Style'].replace("_", " ").title())
    operas = interface.query(query_operas)
    places = interface.query(query_places)
    return info, related_artists, styles, operas, places
```

6.3 Graphic interface

The graphic interface was realized woth the **tkinter** library. Images were added with the intention of creating a more attractive interface but also for completing the information provided to the user. The interface can be started by running the file **gui/homepage.py.** Here are some screenshots of the windows:


ArtHelper
Seek information about styles
Go!
Seek information about artists and architects
Go!
Seek information about artworks
Go!
Seek information about churches
Go!
Figure 1: Homepage

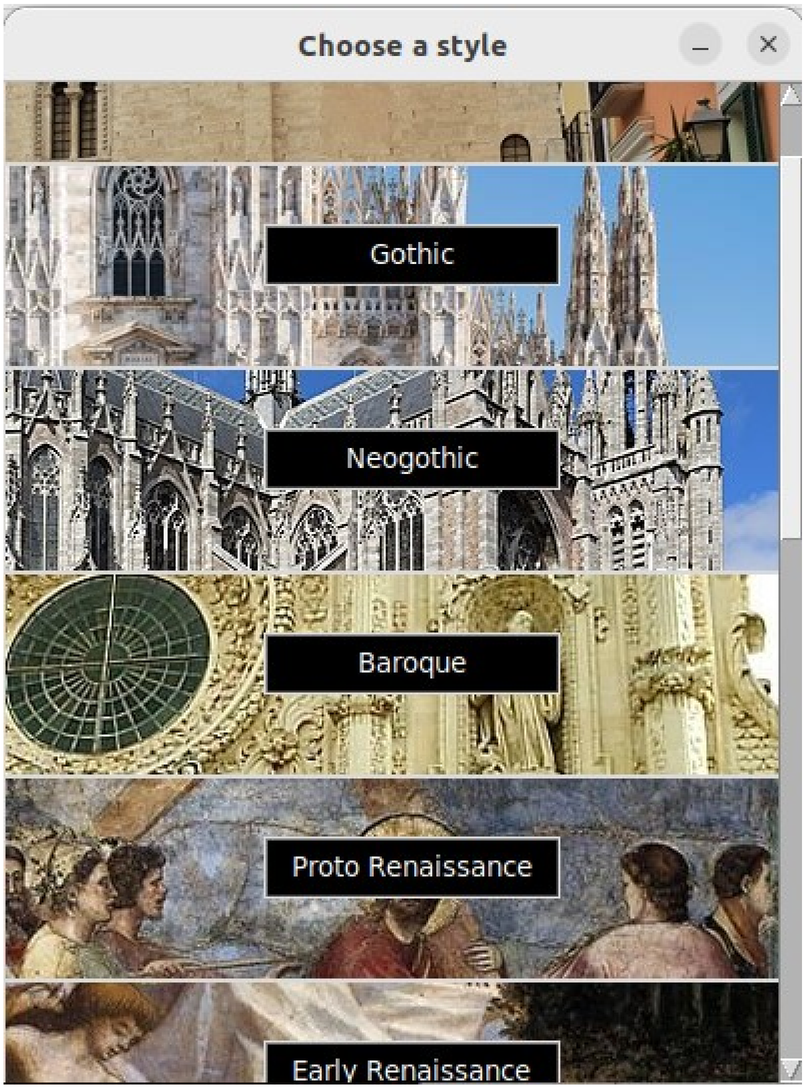
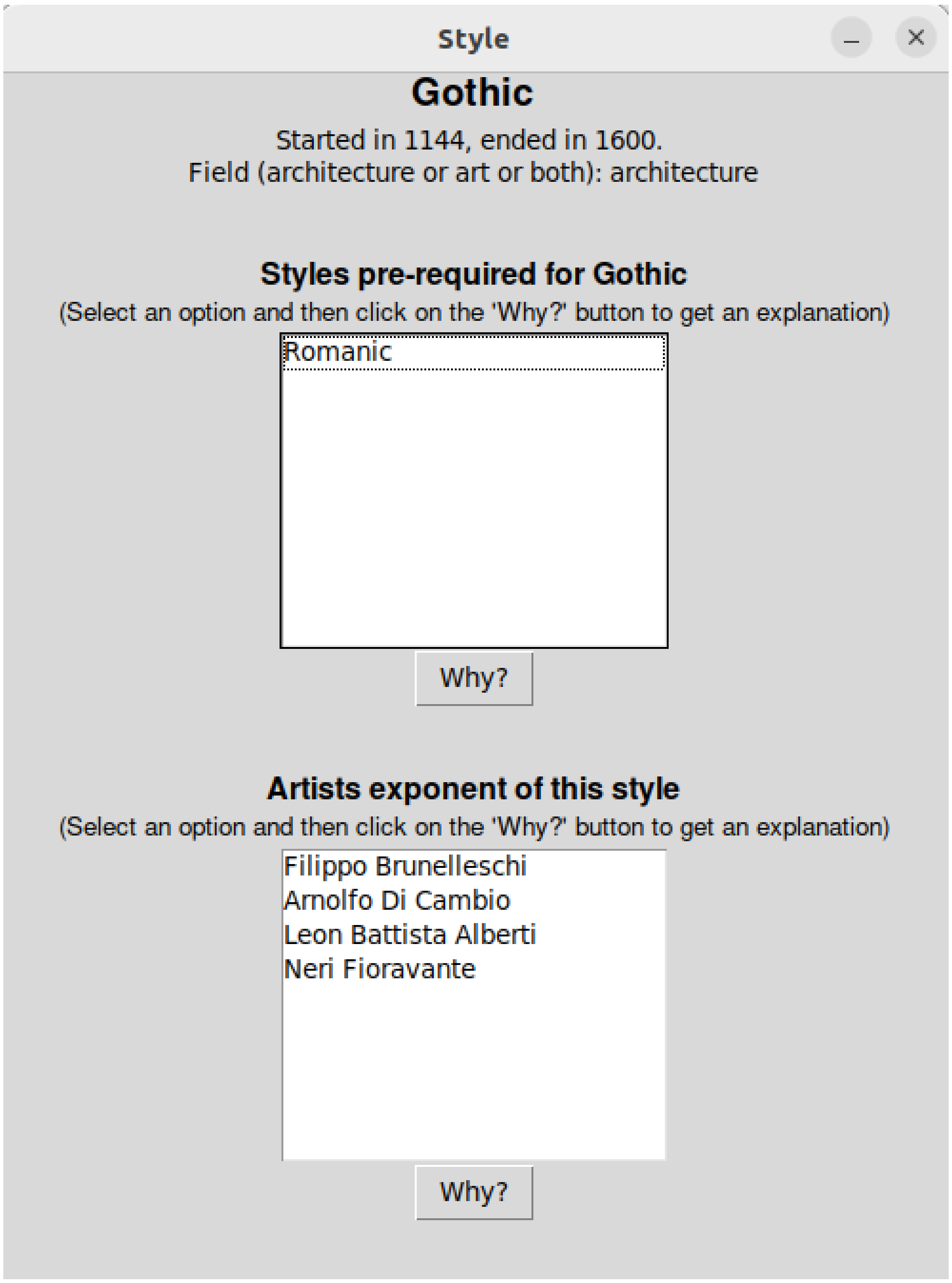


Figure 2: Window where the user can choose a style. Use the scrollbar to see all of them. Click on the button containing the name of a style to get information about it


Style

Gothic

Started in 1144, ended in 1600.
Field (architecture or art or both): architecture

Styles pre-required for Gothic

(Select an option and then click on the 'Why?' button to get an explanation)

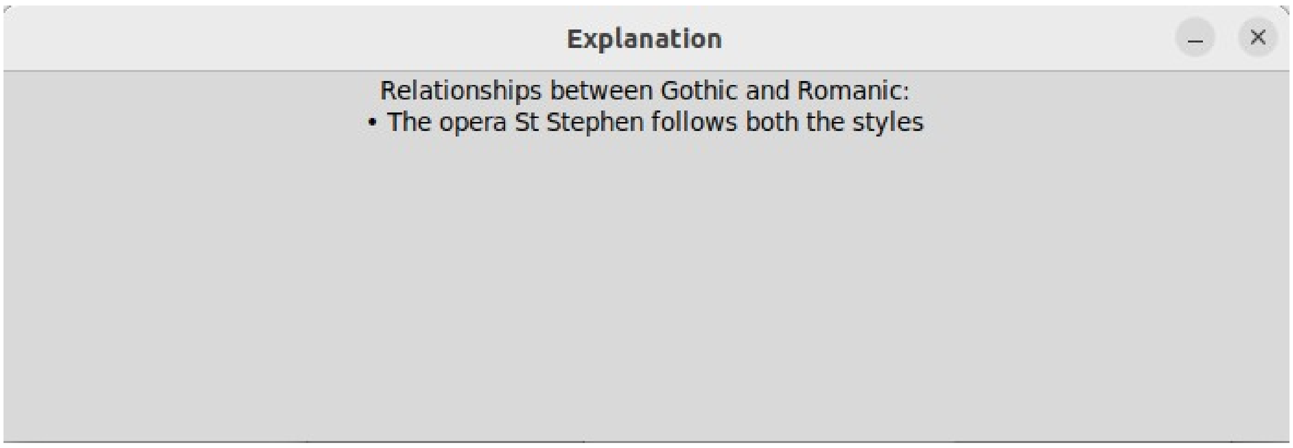
Romanic

Artists exponent of this style

(Select an option and then click on the 'Why?' button to get an explanation)

Filippo Brunelleschi
Arnolfo Di Cambio
Leon Battista Alberti
Neri Fioravante

Figure 3: Window showing all the information about a style


Explanation

Relationships between Gothic and Romanic:
• The opera St Stephen follows both the styles

Figure 4: Explanation window

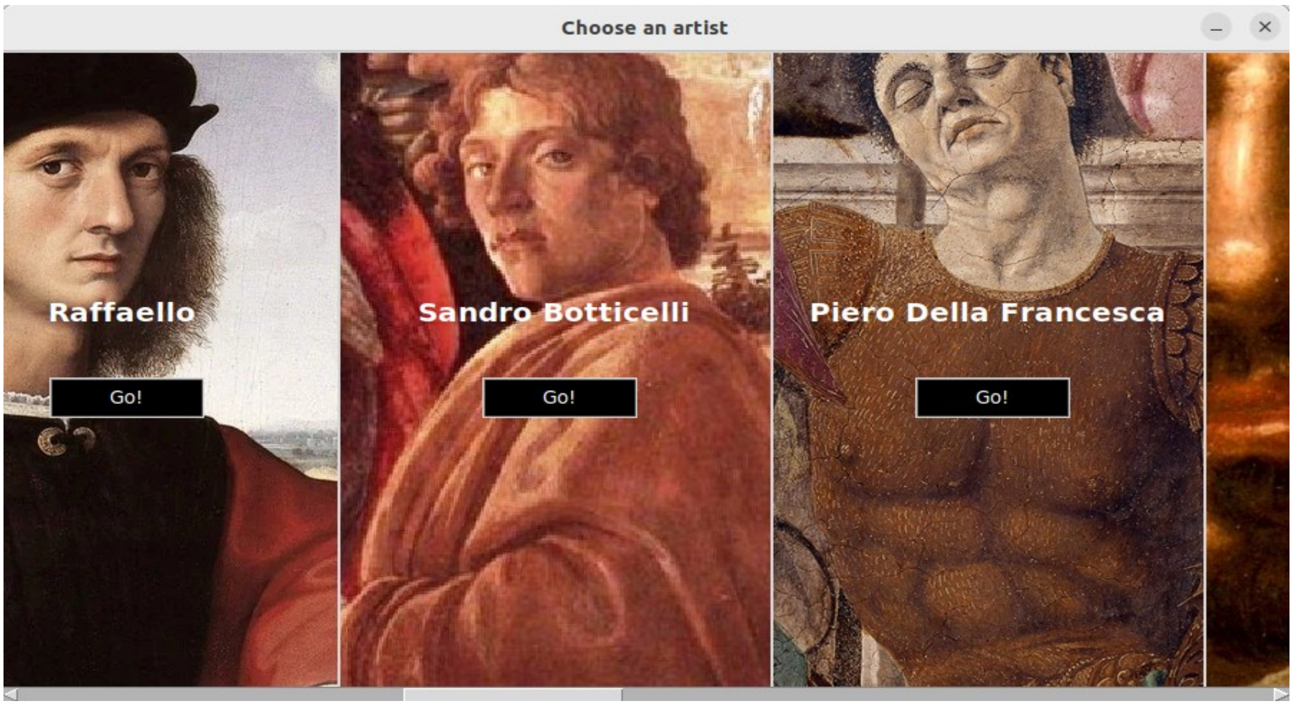
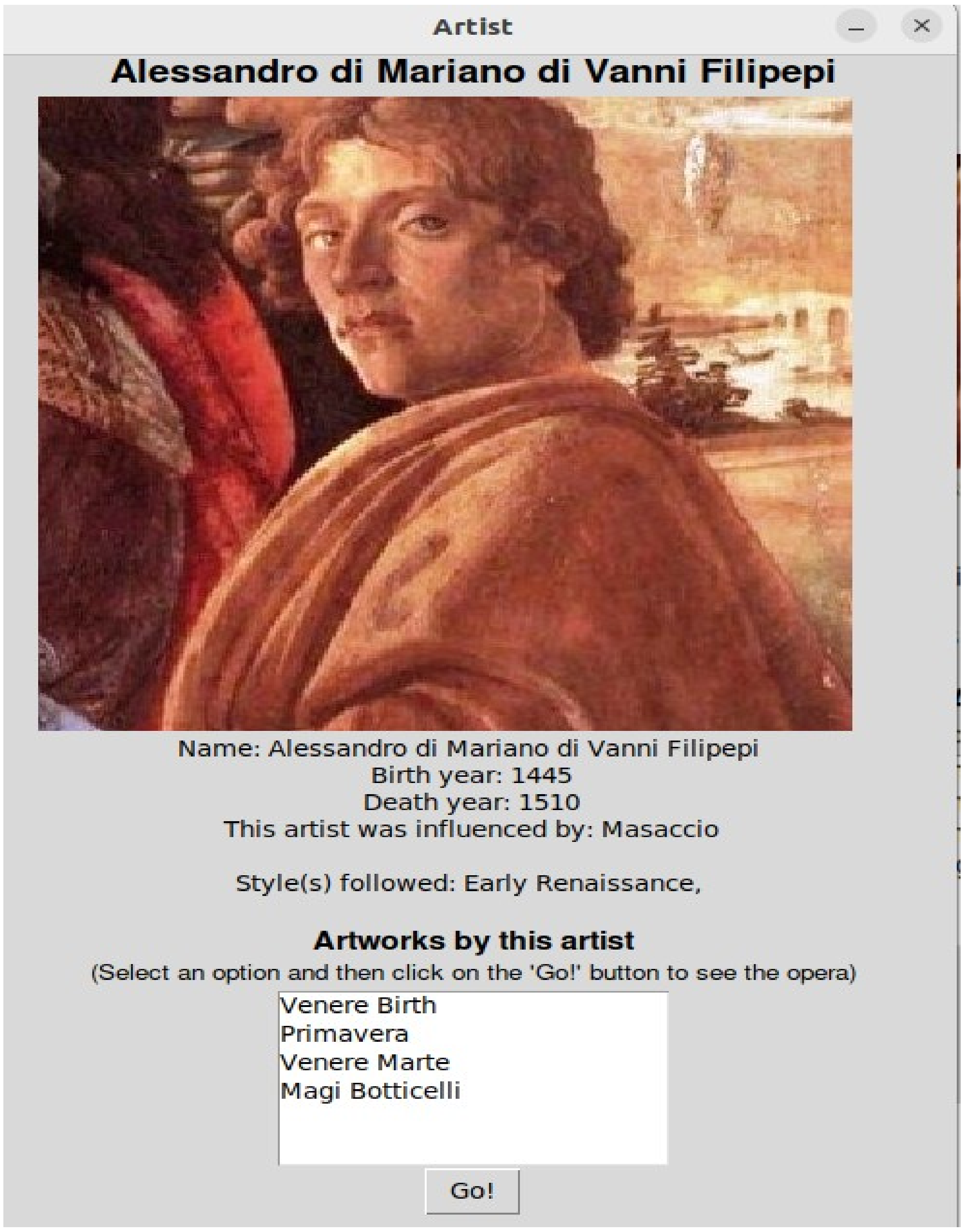


Figure 5: Window for choosing an artist. Use the scrollbar on the bottom to see all of them. Click on the 'Go!' button corresponding to an artist to get information about him



Alessandro di Mariano di Vanni Filipepi

Name: Alessandro di Mariano di Vanni Filipepi
Birth year: 1445
Death year: 1510
This artist was influenced by: Masaccio

Style(s) followed: Early Renaissance,

Artworks by this artist

(Select an option and then click on the 'Go!' button to see the opera)

Venere Birth
Primavera
Venere Marte
Magi Botticelli

Figure 6: Window showing all the information about an artist

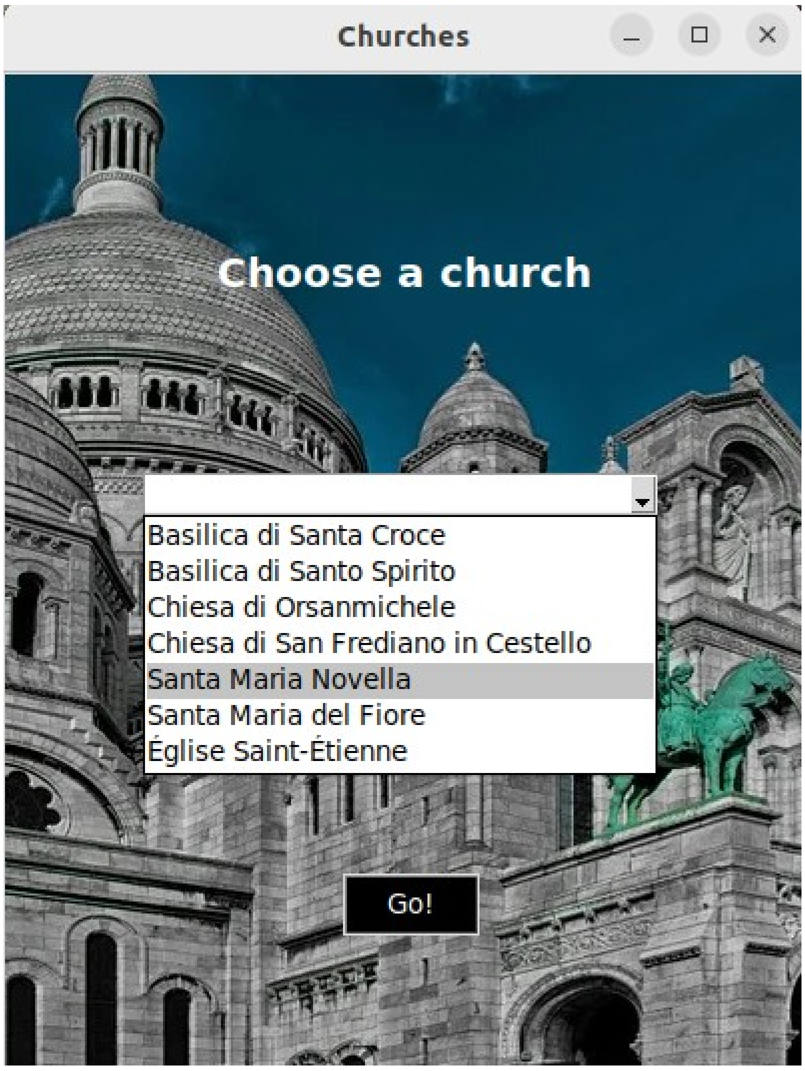
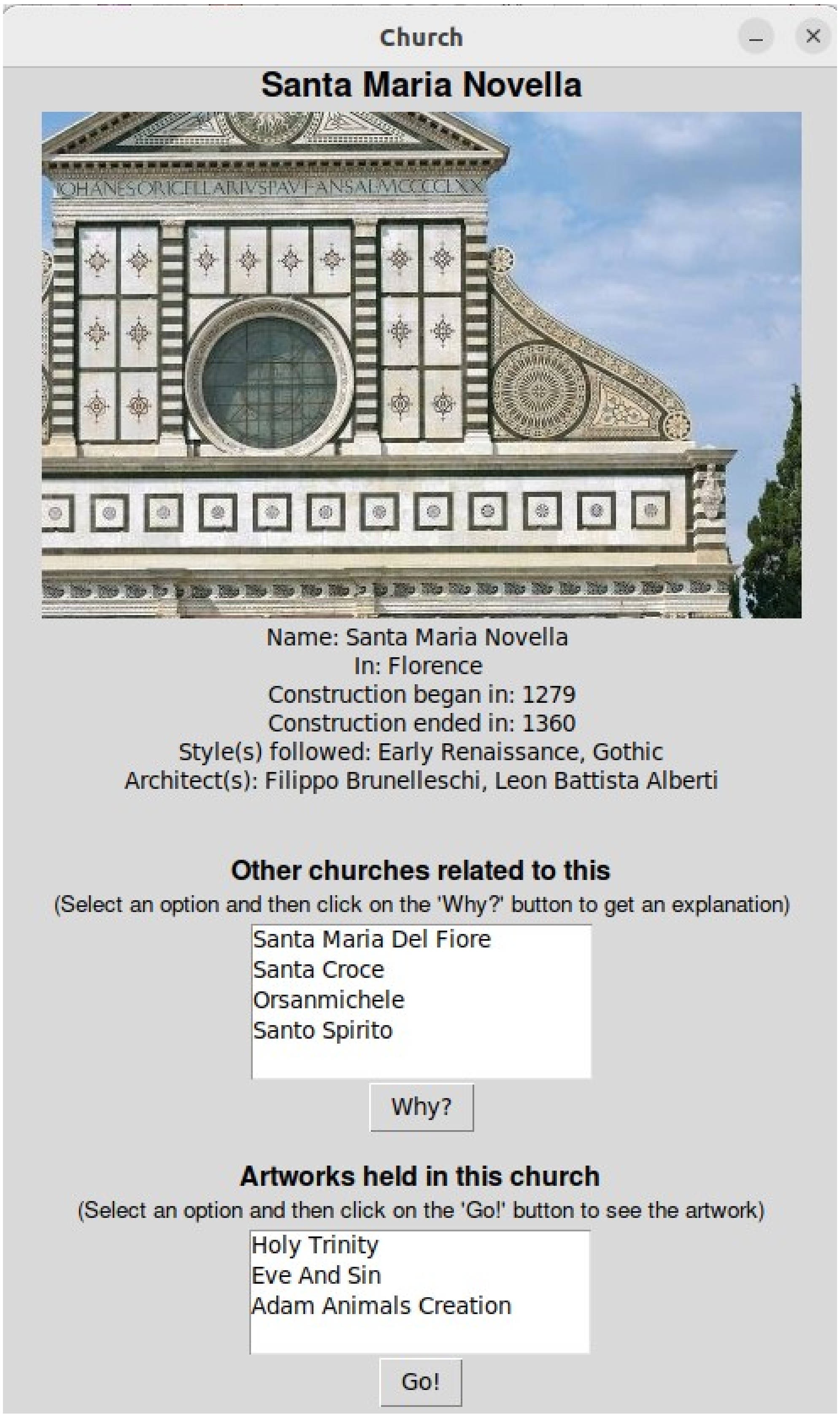


Figure 7: Window for choosing a church. Select the church from the combobox and then click on 'Go!' to get information about it



Church

Santa Maria Novella

Name: Santa Maria Novella
In: Florence
Construction began in: 1279
Construction ended in: 1360
Style(s) followed: Early Renaissance, Gothic
Architect(s): Filippo Brunelleschi, Leon Battista Alberti

Other churches related to this

(Select an option and then click on the 'Why?' button to get an explanation)

Santa Maria Del Fiore
Santa Croce
Orsanmichele
Santo Spirito

Why?

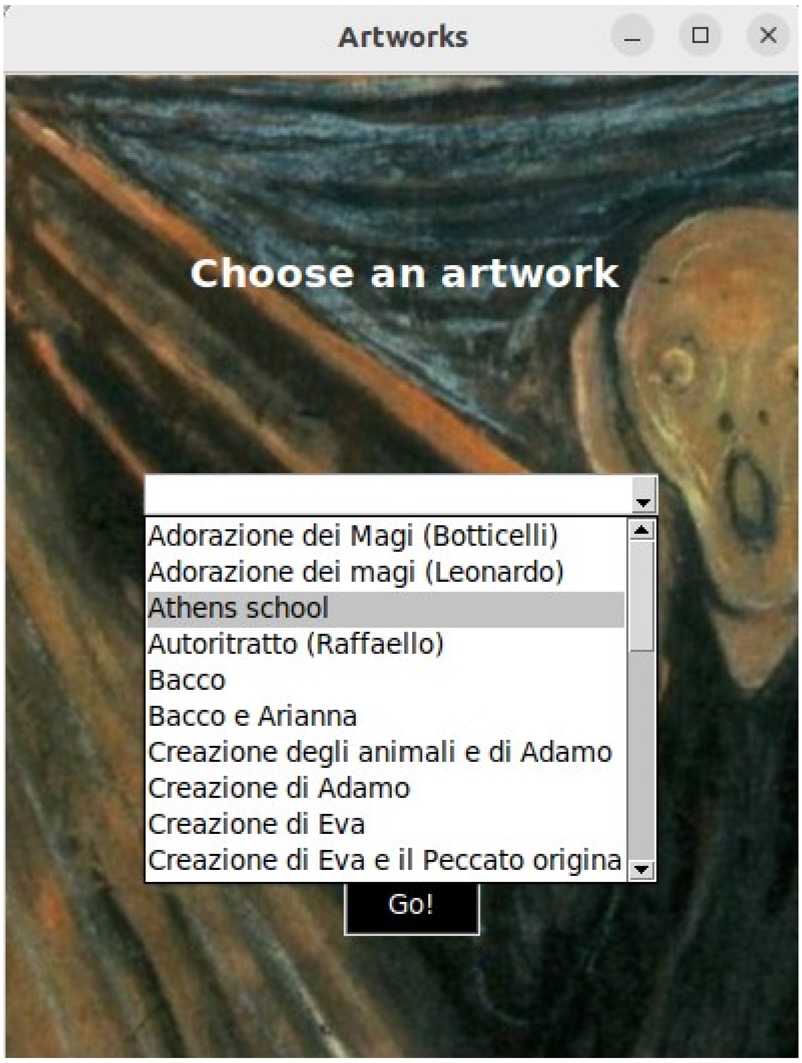
Artworks held in this church

(Select an option and then click on the 'Go!' button to see the artwork)

Holy Trinity
Eve And Sin
Adam Animals Creation

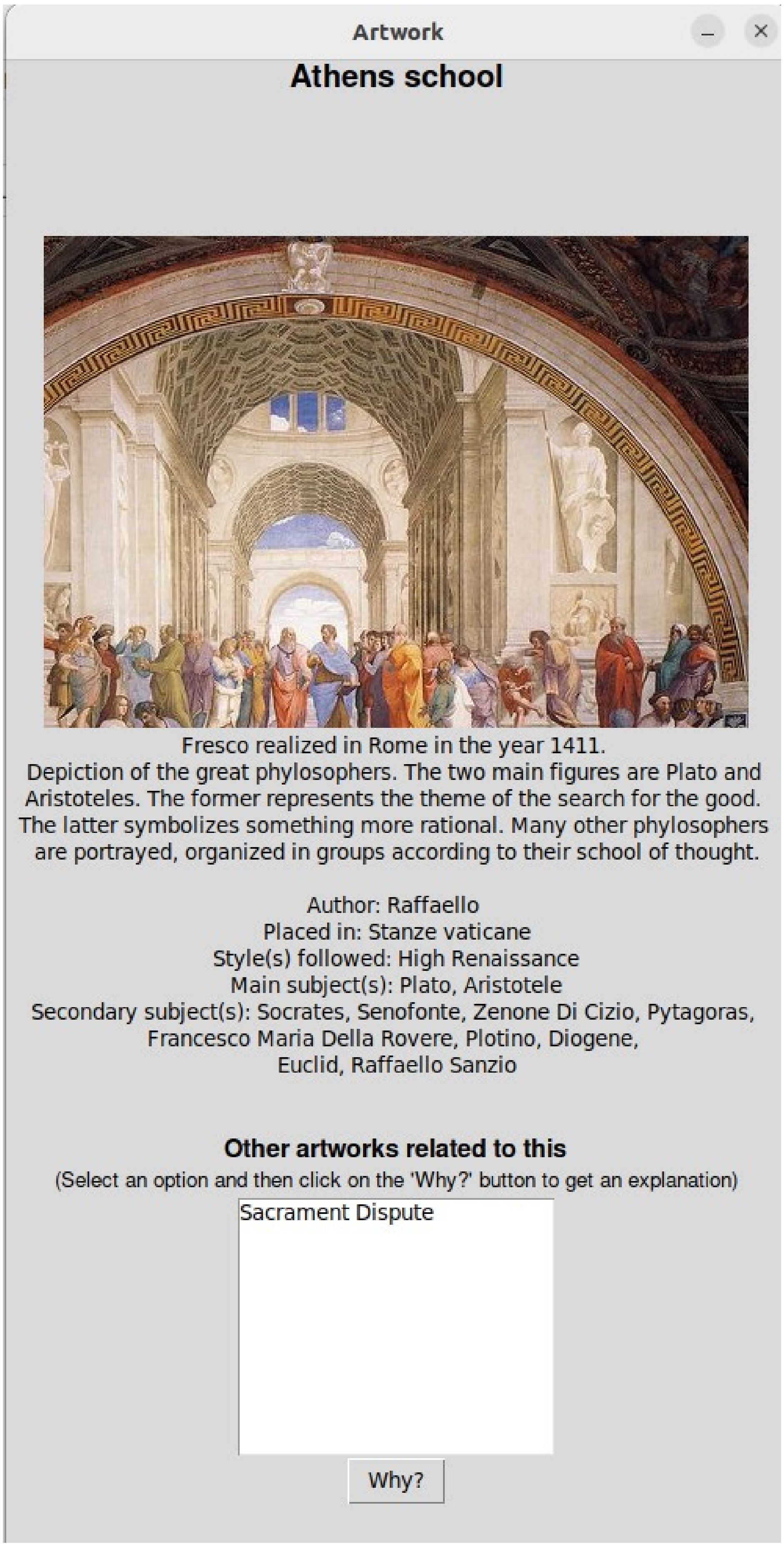
Go!

Figure 8: Window showing information about a church



Adorazione dei Magi (Botticelli)
Adorazione dei magi (Leonardo)
Athens school
Autoritratto (Raffaello)
Bacco
Bacco e Arianna
Creazione degli animali e di Adamo
Creazione di Adamo
Creazione di Eva
Creazione di Eva e il Peccato origina

Figure 9: Window for choosing an artwork



Athens school

Fresco realized in Rome in the year 1411.
Depiction of the great phylosophers. The two main figures are Plato and Aristoteles. The former represents the theme of the search for the good. The latter symbolizes something more rational. Many other phylosophers are portrayed, organized in groups according to their school of thought.

Author: Raffaello
Placed in: Stanze vaticane
Style(s) followed: High Renaissance
Main subject(s): Plato, Aristotele
Secondary subject(s): Socrates, Senofonte, Zenone Di Cizio, Pytagoras, Francesco Maria Della Rovere, Plotino, Diogene, Euclid, Raffaello Sanzio

Other artworks related to this

(Select an option and then click on the 'Why?' button to get an explanation)

Sacrament Dispute

Figure 10: Window showing information about an artwork

7. Conclusions and future works

This report describes ArtHelper a system aimed at helping users in detecting what are the notions that are pre-required for completely understanding an artwork, church, style or artist. There are also an explanation module, that makes ArtHelper able to explain the user why an item is being suggested as pre-requisite for another item, and a Graphic User Interface.

The system could be further expanded with more artists, styles and artworks or even new types of artworks (like sculptures, mosaics, monuments) that could be used as standalone entities with their own requisites, but could also be used for providing even more background information about the items that are already present. These artworks could be manually added by coding them into the knowledge base, or be automatically extracted from an existing knowledge graph like DBPedia or WikiArt. This would require a pre-processing phase aimed at removing all the additional and irrelevant information that would be taken from the graph, and keep only the data useful for the goal of this system.