POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

POLITECNICO
MILANO 1863

Digital Storytelling for People with Intellectual Disabilities: an Intermediate Language for Executable Representations

Advisor: Prof. Licia Sbattella

Co-advisor: Ing. Roberto Tedesco

Master thesis of:

Sara Fiori

Matr. 883760

Academic Year 2019 - 2020

Alla mia famiglia (Laura, Nicola, Paolo, Francesco, Anna),
che mi è sempre e da sempre vicina.
A Luchino,
che mi ha preso per mano.

Acknowledgments

I would like to thank my thesis advisor Prof. Licia Sbattella for the possibility to work on this interesting and meaningful project. I am extremely thankful to Eng. Roberto Tedesco for providing constant support and guidance, although everything.

I sincerely thank the Esagramma association, in particular the educators (Gabriele Rubino, Cecilia Pagliardini, Cristina Bulgheroni and Silvia Spelta) and A, M, U and G, who helped me with the experiments and made me feel at home. Regarding the experiments I would like to tank also the Amicinsieme association, members and volunteers, who are like a family to me.

I would like to remember all the staff and professors of the Polimi Como campus, who guided or inspired me during this master's degree (in particular, Prof. Tedesco, Prof. Di Blas, Emanuela, Prof. Moseneder, Prof. Della Valle, Prof. Brambilla, Prof. Fraternali, Prof. Comai, Prof. Zanero).

I would like also to thank my family, my closest aunts and uncles, the Adriani family, my colleagues, the Salloum family, and my friends, who always helped and supported me.

In conclusion, I wish to present my warmest thanks to Marco who has stood by my side with infinite patience and excellent advice, and who has inspired me to perform better since the first day of my studies at Politecnico di Milano.

Abstract

When looking at the field of applications to create animations and videos with the perspective of someone who wants to tell a simple animated story, we can note that such software tends to be more complex than what would be needed for such a simple need and this, in turn, makes it difficult to successfully use such applications in this field.

One of the most important features that the software mentioned above sometimes is lacking is the possibility to abstract and personalize the main concepts useful for the description of a story. There are specific software that have such features, but they are either too specific (for example only for the creation of comics) or too generic (for instance for the creation of games, animations and similar).

Therefore, with this thesis, we want to address such gap by providing tools to simplify the process to build an application targeted at the creation of animated stories.

A good "one size fits all" solution is difficult to implement in this field; thus, we decided to design an *intermediate language* that abstracts the main concepts useful for the description of the story/animation, being flexible and independent from both the specific application user interface and the execution model. This allows programmers to focus on their specific needs in terms of storytelling, delegating a significant part of the technical details to our solution.

To validate our approach and provide an example of use of Intermediate Language, we also applied it to the design and implementation of an application for storytelling that is easy to use and follows some requirements. The application allows to create and animated stories or life events, using your own imagination and your own voice, and to share or re-watch them at any time. This can be useful for different kinds of use cases, like rehabilitation or playing time for people with intellectual disabilities or children.

The created application starts from a story in a personalized graphical language and leverages the *intermediate language* to generate a Scratch executable file. With tests on the field, we showed that the application can be really useful for the people with intellectual disabilities.

Sommario

Guardando al campo delle applicazioni per creare animazioni o video dal punto di vista di chi necessita di raccontare una semplice storia animata, si può notare come tali software tendano ad essere più complessi di quanto sarebbe necessario e che questo renda difficile utilizzare con successo tali applicazioni in questo campo.

I software sopra menzionati sono, talvolta, privi della possibilità di astrarre e personalizzare i concetti principali utili alla descrizione di una storia. Esistono software o piattaforme che dispongono di tali funzionalità, ma sono o troppo specifici (solo per creare fumetti) o troppo generici (per creare giochi e animazioni complesse).

Con questa tesi vogliamo quindi colmare tale lacuna fornendo strumenti per semplificare il processo di realizzazione di un'applicazione mirata alla creazione di storie animate.

Poiché ci è chiaro quanto realizzare una buona soluzione adatta a tutti gli scenari sia difficile in questo campo, abbiamo deciso di progettare un linguaggio intermedio che astragga i concetti principali utili per la descrizione della storia o animazione, essendo flessibile e indipendente sia dall'interfaccia utente che dal modello di esecuzione. Ciò consente ai programmatori di concentrarsi sulle proprie esigenze specifiche in termini di storytelling, delegando una parte significativa dei dettagli tecnici alla soluzione proposta.

Per confermare la bontà del nostro approccio e fornire un esempio di utilizzo del Linguaggio Intermedio (LI), abbiamo realizzato un software per il "digital storytelling" che sia facile da usare e che rispetti alcuni requisiti preposti. L'applicazione consente di creare e animare storie o eventi quotidiani, utilizzando la propria immaginazione e voce, e di condividerli o rivederli online. Questo può essere utile per diversi casi d'uso, come la riabilitazione o il gioco per persone con disabilità intellettive o bambini.

L'applicazione creata parte da una storia in un linguaggio grafico realizzato ad hoc e sfrutta il LI per generare un file eseguibile dalle piattaforme Scratch. Con i test svolti sul campo, abbiamo dimostrato che l'applicazione può essere davvero utile per le persone con disabilità intellettiva.

1 Introduction

In this Section, we will present the motivations behind the topic for the thesis, and an overview of the aims of our work.

We will also briefly describe the structure of this document.

1.1 Motivation

The evolution that has taken place in recent years in the field of technology has radically changed the way of building and organizing an abundance of information, which has implications both in social and scientific terms, and it has not left the field of storytelling untouched.

Digital Storytelling is a relatively new form of personal expression and art, evolved over the last decade, which uses video, music, picture and narration to create personal short video stories about people's lives, their work and their experiences, shared with peers and online.

When looking at the field of applications to create animations and videos with the perspective of someone who wants to tell a simple animated story, we can note that such software tends to be more complex than what would be needed for such a simple need and this, in turn, makes it difficult to successfully use such application in this field.

For instance, on one hand, tools for presentations (e.g., Microsoft PowerPoint) allow to create animated and interactive sequences of moving objects like texts and images and to personalize the content, but they do not abstract concepts like characters, scenes, backgrounds, dialogs because every page and every object of a page are independent.

On the other hand, video editing tools (e.g., Windows Movie Maker) allow to create sequences of images or videos moving text on it, but they lack the possibility to interact with them, and do not abstract concepts like characters, acts/scenes, backgrounds, dialogs, type of actions etc.

One of the most important features that the software described above is lacking is the possibility to abstract and personalize the main concepts useful for the description of a story. There are specific software or platforms that have such feature, but they are either too specific or too generic.

For instance, tools for the creation of animations, like Scratch (described in Section 2.1.1 and Section 2.3.1), are too generic because they allow the creation of games or complex animations, so they are more powerful, but they add a lot of complexity for our precise purpose.

While tools for the creation of static stories, like Pixton (as seen in Section 2.3.1), are too specific because they only allow to create static story (like comics), so they lack the possibility to describe movements and actions.

1.2 Aims

Therefore, with this thesis, we want to address such gap by providing tools to simplify the process to build an application targeted at the creation of animated stories.

Because we understand that a good “one fits all” solution is difficult to implement in this field, we designed an *intermediate language* that abstracts the main concepts (e.g., definition of the various phases of story, selection of custom images for background and characters, recording of voice or sounds), being flexible and independent from both the specific application user interface and the underlying execution platform. This allows programmers to focus on their specific needs in terms of storytelling, delegating a significant part of the technical details to our solution.

Moreover, employing such abstraction level allows to adapt to new or changed needs more easily, both from a requirements/storytelling perspective and from a technological one, so that the time needed to perform such changes, if needed, can be significantly reduced.

To validate our approach, we also applied it to the design and implementation of an application for storytelling that is intuitive and easy to use even for people with intellectual disabilities.

The application allows to create and animated stories or life events, using your own imagination and your own voice, and to share or re-watch them at any time. This can be useful for different kinds of use cases, like rehabilitation or playing time for people with intellectual disabilities or children.

This application was tested on the field thanks to the collaboration with educators, volunteers, and people with intellectual disabilities from the Esagramma[1] and Amicinsieme[2] associations. During the experiments, the subjects have created stories

[1] Fondazione Sequeri Esagramma Onlus: https://esagramma.net/
[2] Associazione Amicinsieme: http://amicinsiemelipomo.altervista.org/

using their voice and imagination to animate it. This approach can support both educators and their patients, in several ways.

1.3 Outline

This document is structured as follows.

- Section 2 will provide information on relevant works or literature addressing problems related to the one tackled in the thesis.
- In Section 3, the intermediate language will be analyzed in detail, describing basic idea and architecture, with its various components.
- Section 4 will contain the description of the case of study (desktop application).
- In Section 5, the user interface of the application will be presented.
- Section 6 will provide some implementation details of the case of study.
- In Section 7, the results of experiments carried out to evaluate the performance and the aims of the app will be described.
- Section 8 will contain the final comments and directions for possible future evolutions of the work.

2 Background and Related Works

In this Section, we will present relevant works addressing problems related to the one tackled in the thesis.

2.1 Graphical Languages

A general analysis of graphical languages is presented in this Section, mainly focused on Scratch.

2.1.1 Scratch

Scratch is an online community and a programming language made of "coding blocks", from Massachusetts Institute of Technology (MIT) where users (especially young people) can program and share interactive media, such as games and animations, with people from all over the world. The users use Scratch to code their own interactive stories, animations, games, etc. and, in the process, they learn to think creatively, reason systematically, and work collaboratively (Figure 2.9 shows an example of the Scratch tool). The Scratch Wiki[3] contains many pieces of information about the Scratch programming language and in (Maloney, et al., 2010) the authors explore the Scratch programming language and environment.

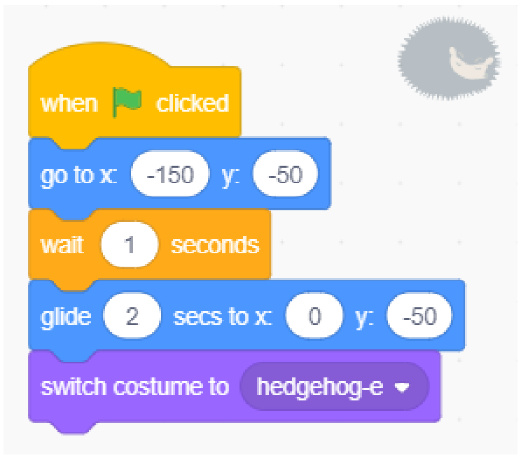
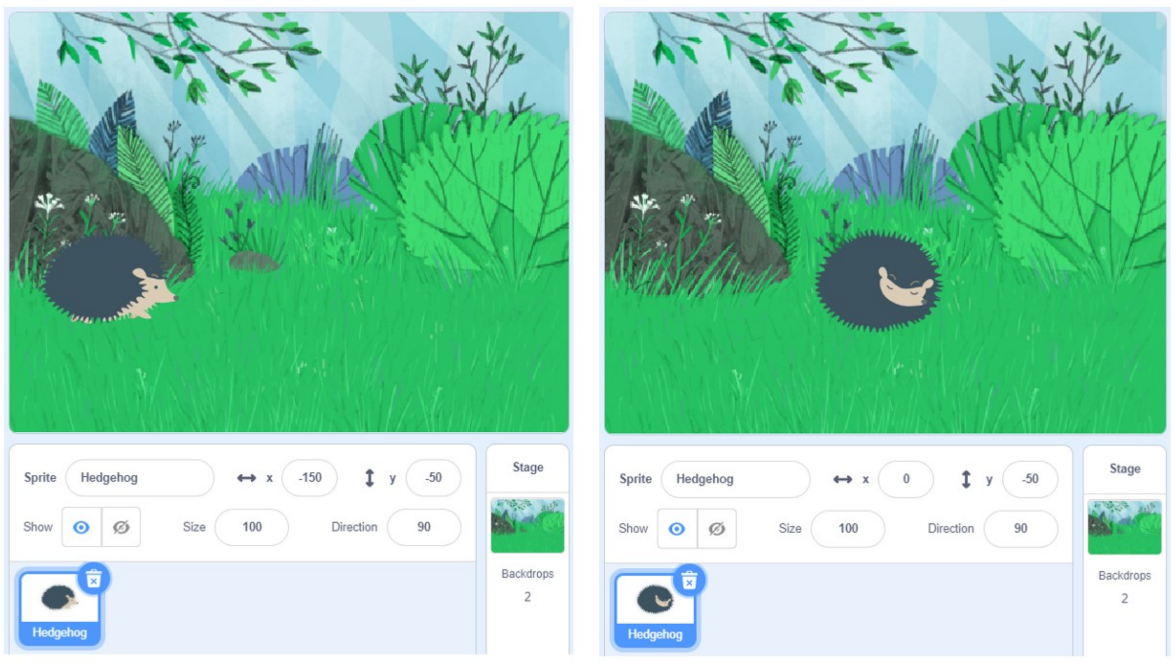

```
when [flag] clicked
go to x: -150 y: -50
wait 1 seconds
glide 2 secs to x: 0 y: -50
switch costume to hedgehog-e ▾
```

Figure 2.1 - Example of Scratch blocks

The Scratch graphical language is made of blocks, that have different colours according to the category they belong to. The categories are: *Motions*, *Looks*, *Sound*, *Events*, *Control*, *Sensing* and *Operators*. In Figure 2.1, an example of the use of Scratch blocks is shown: the first block is an *Event* block, the second and the fourth are two *Motions* blocks, the third is a *Control* block and the last one is a *Looks* block.

[3] Scratch Wiki: https://en.scratch-wiki.info/wiki/Scratch_Wiki_Home

In the right-top corner of Figure 2.1 is also shown the *Sprite* the blocks are associated to (a hedgehog). Sprites are the objects that perform actions in a Scratch project, while *Stage* is the background of the project, they both can have scripts, costumes, and sounds. Users can give instructions to a sprite (such as telling the sprite to move) by snapping blocks together in the scripts area (the Figure 2.1 is a part of the script area of the hedgehog sprite). In the Figure 2.2 we can see the screen of the Scratch project before (left side) and after (right side) the action of the blocks showed in Figure 2.1 are performed.



Sprite Hedgehog x -150 y -50
Show Size 100 Direction 90

Sprite Hedgehog x 0 y -50
Show Size 100 Direction 90

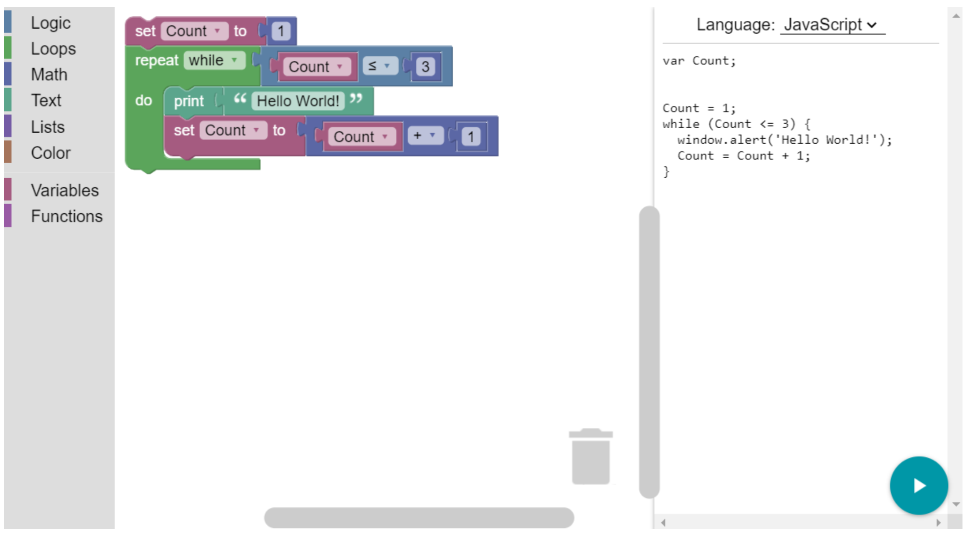
Figure 2.2 - Example of Scratch blocks execution

In (Frädrich, et al., 2020) the authors present a catalogue of “bug patterns” based on common misunderstandings and programming errors in Scratch. In the paper there are three different type of patterns: bugs that are effectively syntax errors, bugs that can occur in any programming language; and bug patterns that are specific to Scratch.

The authors of (Stahlbauer, et al., 2019) describe the Scratch Programming Language with a syntactic model, a semantic model and from the user perspective. In the paper, a formal framework that defines Scratch programs and testing is introduced. The paper also focused on empirical evaluation of the feasibility of Scratch testing and the effectiveness of Whisker (a concrete instantiation of the formal Scratch testing framework) on Scratch projects in the use of the automatic tests for auto-grading.

2.1.2 Tools and literature

Scratch is built on the *Blockly*[4] code base, a Google project. Blockly is a library that adds a visual code editor to web and mobile apps and allows users to apply programming principles without having to worry about syntax. The editor (an example is showed in Figure 2.3) represents coding concepts as interlocking blocks. Similarly to Scratch, there are different types of blocks represented by colors; in Figure 2.3 it is possible to see the different types of blocks on the left side and an example of blocks usage in the middle. It outputs syntactically correct code in a programming language of choice (e.g., JavaScript, Lua, Dart, Python, or PHP, but it can also be customized to generate code in any textual programming language); on the right side of Figure 2.3 there is an example of the JavaScript code corresponding to the stack of blocks.



Language: JavaScript

```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

Figure 2.3 - Blockly example

There is a number of articles about Blockly, we focused more on ones that offer tips or suggestions related to graphical language in general. We mention a few of them below.

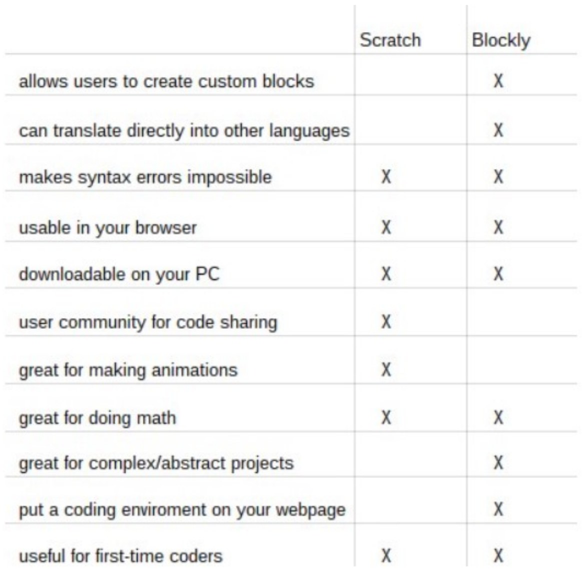
In (Fraser, 2015) the author describes some suggestions and mistakes not to be made and some interesting solutions, applicable to block-based programming in general, collected after a 4-year analysis made on Blockly.

The authors of (Pasternak, et al., 2017) propose the steps to follow to create a block language based on Blockly (but which can also be adapted for a generic graphical language) in terms of: lists of questions to figure out who you are building for and

[4] Blockly web page: https://developers.google.com/blockly

what your overall goals are, how to choose vocabulary and grammar, consistency, default values and testing.

In the same way of Scratch, the Blockly editor, that is it is a ready-made UI for creating a visual language that emits syntactically correct user-generated code, uses interlocking blocks to represent code concepts, including variables, logical expressions, and loops, thus it would be interesting to underline the main differences between the two. In Figure 2.4[5], a comparison between Scratch and Blockly is summarized with a table. There is only a small clarification to add regarding the first row of the table: Scratch also allows to create custom blocks but it is less powerful and less flexible (and it is only possible to create a custom block for a sprite/stage at a time, not shared between them) than the one in Blockly, while Blockly offers a more powerful tool that permits the creation of a totally custom blocks: Block Factory. We can notice that Blockly is more focused on the translation of the projects directly into other languages, while Scratch is more focused in concrete projects with good animations and effects.



	Scratch	Blockly
allows users to create custom blocks		X
can translate directly into other languages		X
makes syntax errors impossible	X	X
usable in your browser	X	X
downloadable on your PC	X	X
user community for code sharing	X	
great for making animations	X	
great for doing math	X	X
great for complex/abstract projects		X
put a coding enviroment on your webpage		X
useful for first-time coders	X	X

Figure 2.4 - Scratch vs Blocky

In (Merwe, et al., 2019) the authors propose a conceptual framework of research on visual language specification languages, developed from the literature review and various studies in this field, based on the current research on specification languages. In the paper the authors also focused on how the conceptual framework can be used for different purposes, like creating comprehensive lists of features that can be used to characterize or evaluate specification languages.

[5] Taken from https://www.robotlab.com/blog/blockly-vs-scratch-whats-best-for-me

A concise description of the conceptual framework of research on visual language specification languages is shown in Figure 2.5, that was taken from the paper.

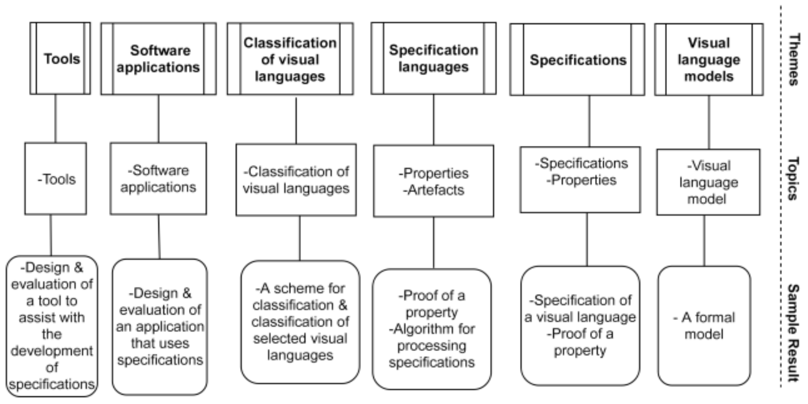

Tools
Software applications
Classification of visual languages
Specification languages
Specifications
Visual language models
Themes
-Tools
-Software applications
-Classification of visual languages
-Properties
-Artefacts
-Specifications
-Properties
-Visual language model
Topics
-Design & evaluation of a tool to assist with the development of specifications
-Design & evaluation of an application that uses specifications
-A scheme for classification & classification of selected visual languages
-Proof of a property
-Algorithm for processing specifications
-Specification of a visual language
-Proof of a property
- A formal model
Sample Result

Figure 2.5 - A summary of the conceptual framework

All six themes of the framework are focused on visual language specification languages. We briefly describe them following the order of Figure 2.5: (1) the theme “Tools” focuses on tools to assist in the development of specifications; (2) “Software applications” focuses on utilization of specifications generated by the specification language; (3) “Classification of visual languages” focuses on classifying visual languages using properties of the selected specification language; (4) “Specification languages” has as main topics of research properties and artefacts of a specification language; (5) “Specifications” centers around the development of specifications for visual languages and their properties; (6) “Visual language models” focuses on the visual language models supported by the specification language.

In (Dann, et al., 2012) the *Alice* programming language and the translation from the Alice 3 abstract syntax tree into Java code are presented. Alice is an object-based educational programming language with an integrated development environment.

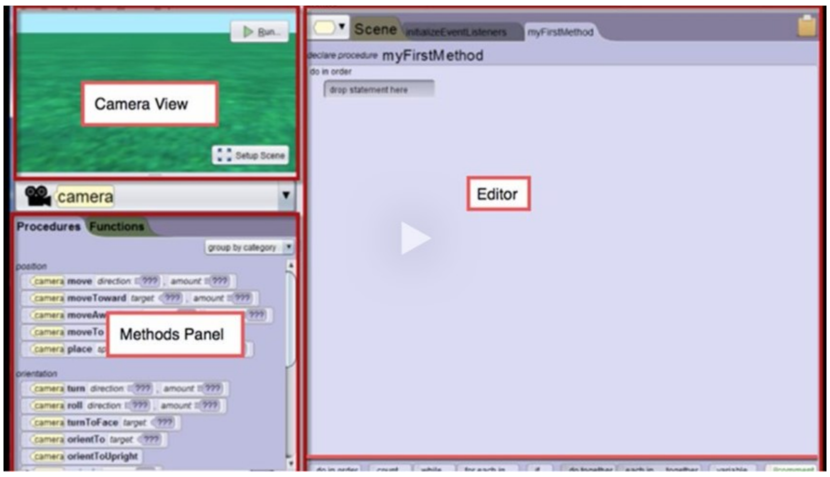


Figure 2.6 - Example of Alice Code Editor

The Figure 2.6 (taken from Alice web site[6]) shows an example of the *Alice Code Editor*, where the user can build a program to create animations or games. The *Camera View* displays the Scene that has been built. In the *Methods Panel* each block represents a method of the object currently selected that can be used in the Editor. A method is an action performed on or by an object (for instance animal, person, prop etc.). The Methods are divided in two categories: procedures (perform an action), and Functions (ask a question or compute a value). In the *Editor* panel, method and control blocks are dragged and dropped into place for the creation of program instructions (the script for the animation).

Alice uses a drag and drop environment to create computer animations using 3D models. In the paper Alice is also compared with Greenfoot[7], that is an interesting interactive environment that enables students to develop 2D graphic applications based on Java.

2.1.3 Remarks

We based our work mainly on Scratch, taking inspiration from it and using it as a target language for the case study. We choose Scratch because it is very good at making animations, easy to use and it contains all the features that we need for the target language. All the papers related to Scratch and many of the contents of the Scratch web site have been useful for this thesis, in particular for the characterization and the translation to Scratch files in the application, and also to inspire concepts and ideas that are present in the intermediate language.

The other papers were also useful for the creation of the graphical language of the application and the intermediate language design; Indeed, they provided tips and suggestions for the creation of the graphical language of our case study and for the interaction with Scratch.

[6] Alice official web site: https://www.alice.org/

[7] More information about Greenfoot can be found here: https://www.greenfoot.org/home

2.2 Describing processes: BPMN

We decided to take inspiration from the BPMN[8] 2.0 language, that has been widely adopted in the recent years as one of the standard languages for visual representation of business processes, for the description and an example of the graphical representation of our Intermediate Language.

2.2.1 BPMN

The Business Process Model and Notation (BPMN) standard allows the modelling of business processes involving one or more organizations, providing a graphical notation to specify business processes in a Business Process Diagram (BPD).

The BPMN is basically a derivation of the formalism of the flow charts but with some additions and modifications that allow to overcome some limits in the modeling of business processes. Its primary goal is to create a standardized bridge for the gap between process design and process implementation in order to entirely automate the business process management lifecycle, but it can be used also to describe other type of processes. In Figure 2.7 (taken from the web site[9] suggested in "BPMN drawing examples" section of the BPMN web site[8]), an example of an Amazon order fulfillment process is shown, that describes a hypothetical process to order an item from Amazon (from the credit card company, customer's, Amazon's and carrier's perspective).

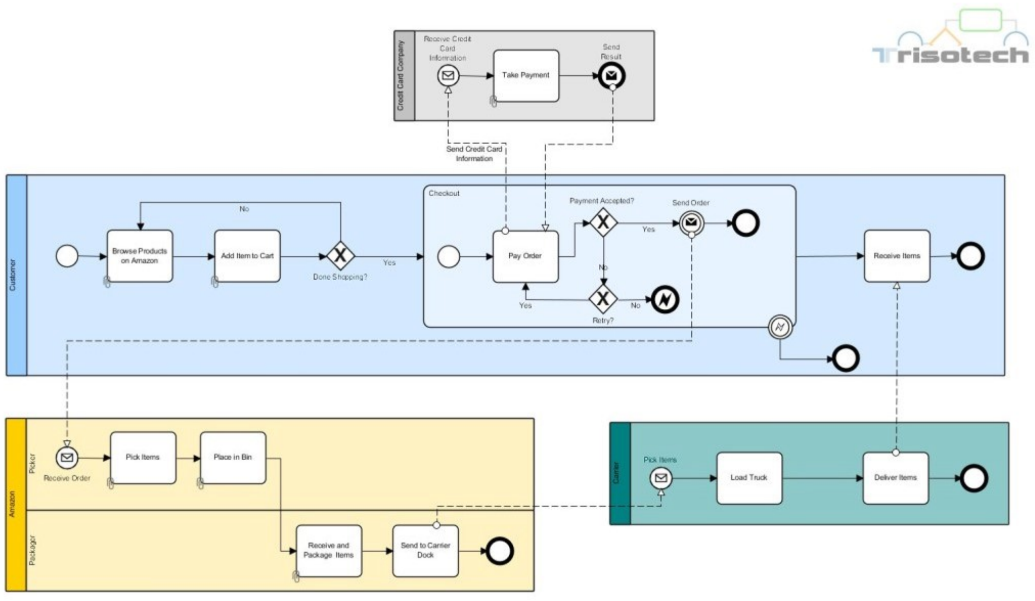


Figure 2.7 - Example of BPMN 2.0

[8] Business Process Model and Notation: https://www.bpmn.org/
[9] https://www.businessprocessincubator.com/content/amazon-fulfillment/

2.2.2 Syntax and semantic

Since the syntax of our Intermediate language has been inspired by BPMN, for the formal description of its syntax and semantic we took inspiration from other related research, and we modified them for our needs.

In (Tiezzi, et al., 2018) the authors describe the BPMN collaborations syntax and semantic. They define the syntax in BNF-style[10], a metasyntax suitable for defining formal operational semantics. Their main effort was defining the operational semantics by relying on the notion of Labeled Transition System (LTS), keeping the formalization easy to understand and effectively exploitable. In the paper they use the formalization of BPMN for the verification of properties of interest for collaboration models expressed in terms of LTL[11] formulae (as noted in the paper, an LTL model checker already exists).

The authors of (Tan, et al., 2017) started from the idea, proposed in (Ouyang, et al., 2007) and (Ouyang, et al., 2008), to map a BPMN model into a Petri net (Figure 2.8 shows an example of Petri net taken from the latter paper) and, considering that as good solution, they decided to expand it. For the authors there are two main advantages of using Petri nets: (1) it is not just a graphical representation but also a mathematical modeling language that has precise execution semantics, and (2) decades of research in Petri nets (PN) offer acknowledged theories to support analysis of Petri-net specified systems.

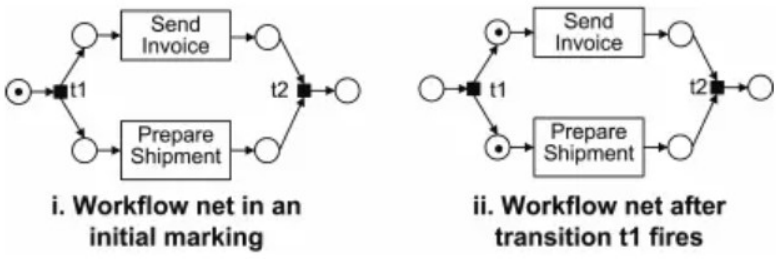

Send Invoice
Prepare Shipment
t1
t2
i. Workflow net in an initial marking
Send Invoice
Prepare Shipment
t1
t2
ii. Workflow net after transition t1 fires

Figure 2.8 - Sample of workflow net in two different states

Petri net is a directed bipartite graph with place and transitions, represented by circles and rectangles, respectively (see Figure 2.8). Places and transitions are connected with directed arcs. The transitions can have an input place (the place that connects to the transition) and an output place (the place that is connected from a transition). Tokens, represented by dots, are located in places in Petri Nets. Each transition can fire only when there is at least one token for each input place of the transition. When a transition fires, it removes a token from each input place of the transition and puts a token to each output place of the transition. A marking of a PN

[10] Backus-Naur Form
[11] Linear temporal logic

is given by a vector that represents the number of tokens for all the places. The one described here is one of the most commonly used definitions of Petri Nets, but there exist many possible variations, involving different kinds of tokens (e.g., coloured tokens), transition fire rules, etc.

2.2.3 Remarks

The syntax definition of the intermediate language is based on the work made in (Ouyang, et al., 2007) and obviously adapted to our needs because our Intermediate Language takes inspiration and concepts from BPMN even if, it is important to underline, it is not a BPMN model.

We define also a semantic for our intermediate language because, as part of our future works in the field, we would like to create a model-checking algorithm that permits to check if all the properties that the Intermediate Language needs to fulfil (described with the syntax) are respected. For the definition of the semantic we follow the suggestions and ideas of (Tan, et al., 2017) and (Ouyang, et al., 2008) because, as the authors have pointed out, Petri nets have well-established theories to support analysis and verification of the properties, while the Model Checking algorithm for LTL, suggested as semantic definition for BPMN from the other authors, is particularly complex and therefore little used.

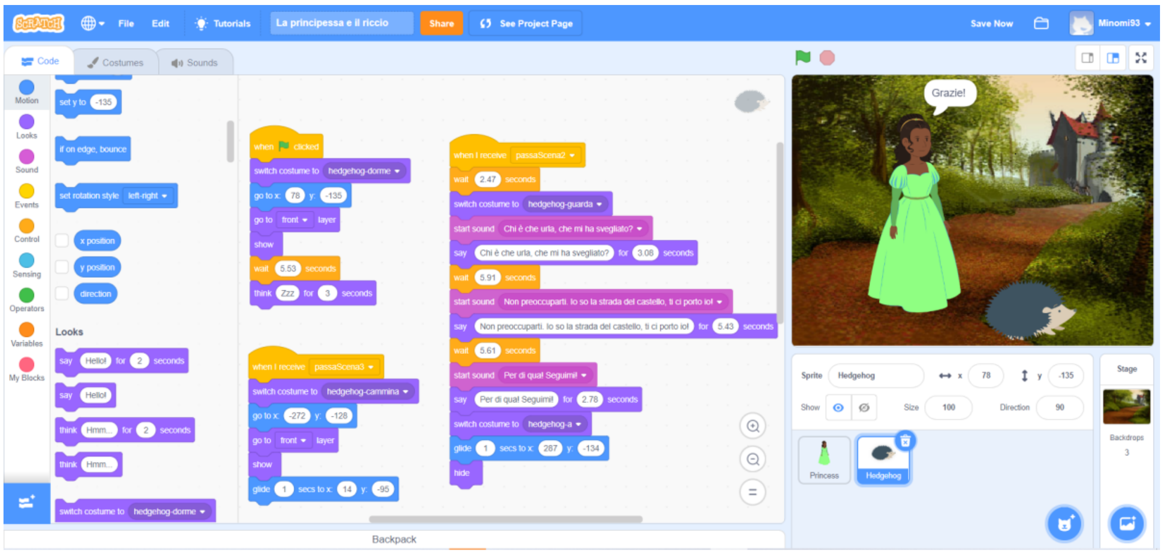
2.3 People with intellectual disability and Storytelling

In this Section we present a brief discussion about storytelling from both the point of view of the already available tools and the one of the relevant literature on the field.

2.3.1 Tools

The existing software for storytelling are basically of two types: the ones that permit to create *animated* stories and the ones that permit to create "*static*" stories. We decided to focus on software that let the user personalize a lot the story and record voices and thus selected the following examples for the two types: *Scratch* for the animated story, and *Pixton* for the static ones.

Scratch, introduced in Section 2.1.1, is a powerful language to create games and animations. As shown in Figure 2.9, leveraging the Scratch Editor, the user can create the content of the work as sets of stacked blocks. The colour of the blocks changes with their meaning, and they can be inserted in a sort of blank sheet; there is one sheet for every subject (in the example below there are two subjects) and one for the Stage. The user can add every type of subject in the project selecting from existing ones or choosing from images on their computer; the same holds for sounds and stage images.



```
when [flag] clicked
switch costume to hedgehog-dorme
go to x: 78 y: -135
go to front layer
show
wait 5.53 seconds
think Zzz for 3 seconds

when I receive passaScena3
switch costume to hedgehog-cammina
go to x: -272 y: -128
go to front layer
show
glide 1 secs to x: 14 y: -95

when I receive passaScena2
wait 2.47 seconds
switch costume to hedgehog-guarda
start sound Chi è che urla, che mi ha svegliato?
say Chi è che urla, che mi ha svegliato? for 3.08 seconds
wait 5.91 seconds
start sound Non preoccuparti. Io so la strada del castello, ti ci porto io!
say Non preoccuparti. Io so la strada del castello, ti ci porto io! for 5.43 seconds
wait 5.61 seconds
start sound Per di qua! Seguimi!
say Per di qua! Seguimi! for 2.78 seconds
switch costume to hedgehog-a
glide 1 secs to x: 287 y: -134
hide
```

Figure 2.9 - Scratch online editor

With Scratch, the user can create a totally personalized story, selecting existing characters and background, adding new images, recording voice and animating everything.

Pixton[12] is an online platform for creating comic stories using pre-designed and widely modifiable models and characters. With Pixton, it is possible to create comic stories, choosing in detail the appearance of the characters, the background and other settings, and inserting the lines of the characters, also with the possibility to record the voice for each line. There is also the possibility to insert photos from the computer.

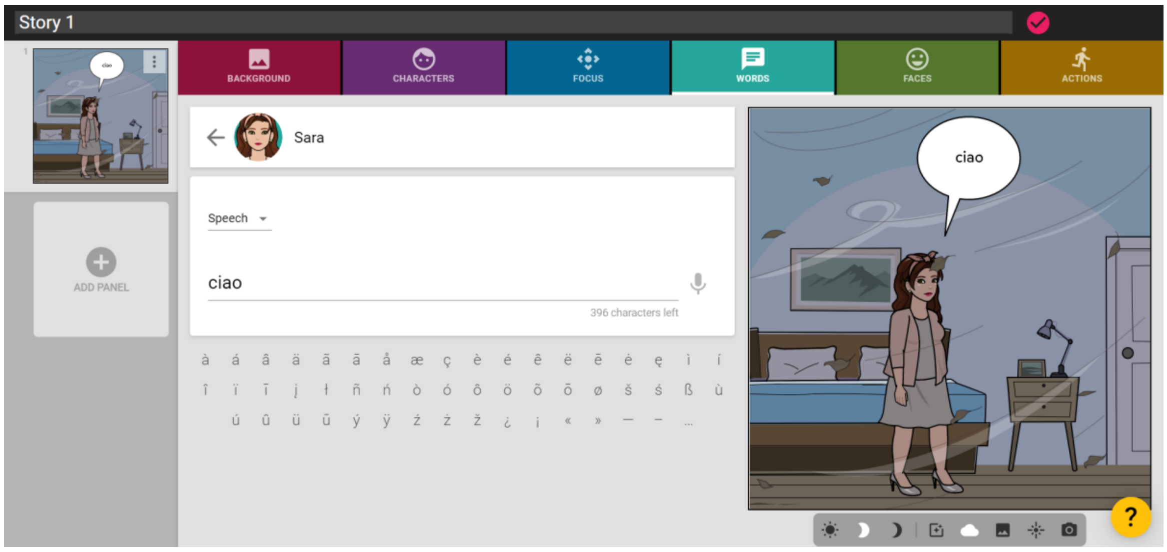


Figure 2.10 - Pixton online editor

2.3.2 Relevant literature

In (Saridaki, et al., 2018), the authors describe the importance of digital storytelling for people intellectual disabilities based on the literature in this field over the past 25 years and from a series of workshops.

In the paper it is underlined that literature recognizes storytelling as a powerful educational tool and that there are many documented case studies and reported positive results of digital storytelling "regarding skills development, self-image perception and personal improvement, strengthening group communication and developing creativity and expression". According to the authors, literature also recognizes the positive outcomes of digital storytelling on people with intellectual disabilities, especially regarding educational skills and personal empowerment.

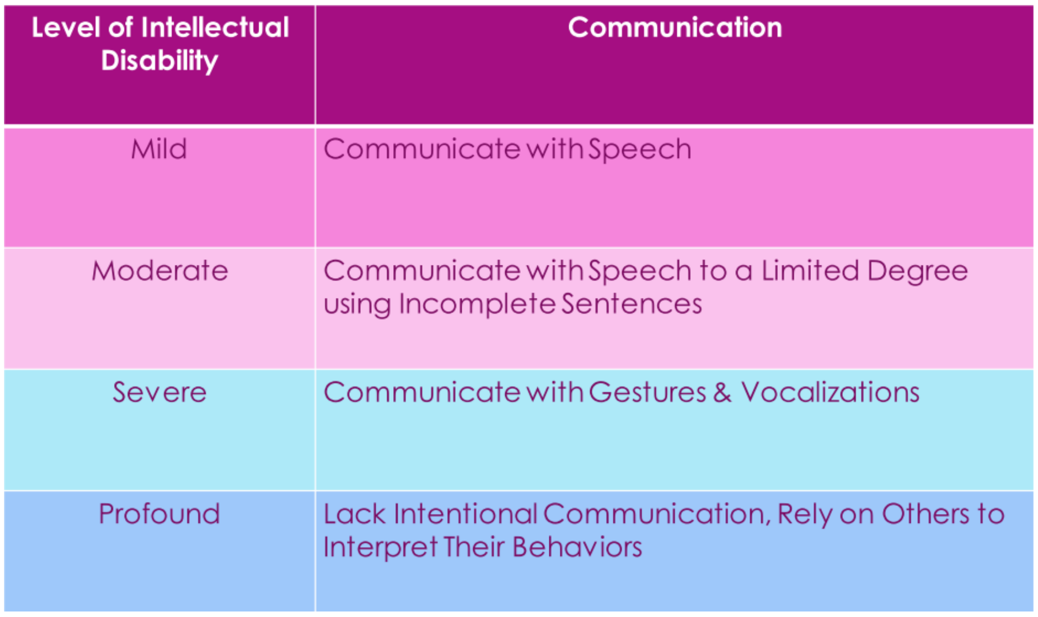
The authors thus conclude that: "Digital Storytelling could be a valuable tool of empowerment to storytellers with intellectual disability, allowing them to use multimedia technologies as an extension of their abilities, skills and intents of

[12] Pixton website and online editor: https://www.pixton.com/

expression. They could be also used in contexts of social projects with people with and without disabilities.”.

In (Tanaka, et al., 2017) the author collects information from literature which falls in line with the findings of this thesis: for people with social communication difficulties, with extreme examples such as autism spectrum disorders, it may be easier to use computers than to directly interact with a human trainer. The advantages of using computers to work with them have been underlined by the article as the following: 1) they prefer computerized environments that are predictable, consistent, and free from social demands; 2) the possibility to work at their own speed and level of understanding; 3) the exercise (like voice recording) can be repeated until the goal is achieved without any pressure; 4) concentration, interest and motivation can be maintained with computerized rewards; and 5) they also show good, and sometimes superior, “systemizing” skills.

The communication abilities in intellectual disability are often an interesting and difficult area where to work from an education point of view. Depending on the level of the intellectual disability of each people the communication skills can be poor or, in the worst cases, almost absent. A summary of the communication abilities in the different levels of intellectual disability are illustrated in the Figure 2.11, taken from (Tahira, 2020).



Level of Intellectual Disability	Communication
Mild	Communicate with Speech
Moderate	Communicate with Speech to a Limited Degree using Incomplete Sentences
Severe	Communicate with Gestures & Vocalizations
Profound	Lack Intentional Communication, Rely on Others to Interpret Their Behaviors

Figure 2.11 - Communication abilities in intellectual disability

In (Garzotto, et al., 2010), the authors designed and implemented low-cost and easily customizable novel learning experiences, combining the visual communication paradigm of Augmented Alternative Communication with multimedia tangible technology. The paper presents the approach and its application in a real school context, highlighting the benefits for both children and children with disabilities. Before the description of the experiments, the authors point out and describe the educational goals and the requirements in terms of technology and children's user experience. The main educational goals presented are: cognitive benefits and affective benefits. The requirements on children's user experience mainly focus on this fields: each child is unique, tangible interaction, use of familiar situated material, consolidation, engagement, concentration and timing. Easy customization (open, highly flexible and adaptable) and low cost are the main requirements on technology discussed.

2.3.3 Remarks

Since Scratch, and similar tools, has been designed to be used for much more complex tasks than just creating a story, using it for such a purpose is not very simple and requires a significant amount of knowledge and time. On the other side, Pixton and similar tools are easier to use but not enough intuitive and, the lack of animations, they lose much of their potential for people with intellectual disability or children.

The relevant literature are all really clear about the needs for people with intellectual disabilities, and the importance of digital storytelling and personalized tools on computers, in the educational field. All the papers studied were really interesting and significant for the conception and the design of the application.

2.4 Discussion

From the information reported in Section 2, it is emerges that it would be really useful, in different educational contexts, to have tools to simplify the process to design applications targeted at the creation of animated stories or videos, that do not necessarily offer complex animations but that allow deep customization of the story, are easy to use and not "boring" (i.e., providing the possibility of adding customized images, photos, drawings etc.).

To be more specific, we think that one of the main features that such tools should provide is flexibility both in the kind of different user interfaces that can be offered to the end user and in terms of the means used to actually make the stories "executable" on computers. This would allow to accommodate the needs of as many users as possible and make the tools usable in educational paths with different needs.

To obtain such a result, we designed an *intermediate language* that abstracts the main concepts (e.g., definition of the various phases of story, selection of custom images for background and characters, recording of voice or sounds), being flexible and independent not only from the execution platform (target) but also from the specific application user interface (source).

This should allow programmers to focus on their specific needs in terms of storytelling, delegating a significant part of the technical details to our solution.
To facilitate this process even more from the programmers' point of view, the intermediate language has to be easy to use and understand, and for this reason we considered useful to build its components on top of existing ideas and languages and to provide an example of graphic representation through BPMN that allows to describe the processes in a "human readable" way.

3 Intermediate Language

Since we are dealing with storytelling, presentation or animations creation, we will try to consider all and only the concepts related to those categories. Therefore, the Intermediate Language is not usable, in most cases, for other purposes.

In the next Sections we will illustrate the ideas and the architecture of the designed solution for the translation, focusing on the definition of the the Intermediate Language (IL), its components and data model, and we will complete the description sketching out a syntax and semantic for the IL.

The case study presented in the chapter 4 is an example of the use of the Intermediate Language described in this Section.

3.1 Introduction

There are many software and tools that allow to create and play video, animations, stories, presentations, but, usually, they are either too complex for what the user needs to do or not powerful enough. Therefore, any programmer who wants to create a simplification or improvement of these tools must create a new visual programming language (source language) based on their needs and then translate it in an existing executable one (target language) – see Figure 3.1.

We would like to provide a supporting intermediate model for their work on translation, that can make the translation process easier and clearer, because it is done in smaller steps, and that is flexible, because in case of changes in the source or target language the programmer can work to modify only one of the two side (e.g., from S to IL or from IL to T).

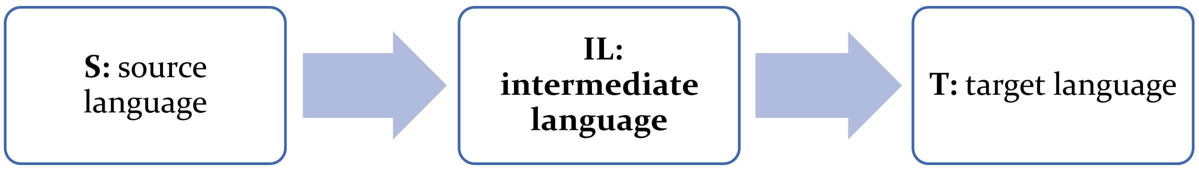


Figure 3.1 - Steps for the translation

The approach that we designed can be used in different fields and allows to make the translation from the customized language, and the translation to executable representation, independent. For instance, thinking about the fact that the project could change: having an intermediate language would allow to change the source language much more easily (like adding/removing new concepts or information, or creating a totally new language) , as well as to change the target language, for example due to a different need in the final result (e.g., we want a video, no more a Scratch file) or due to new version of this one(like, e.g., moving from Scratch 3 to Scratch 4).

In the next Sections we will describe in detail the intermediate language created.

3.2 Idea and features

What we need is a language “somewhere in the middle”, that is neither specific to a given scope, thus being adaptable only to some specific contexts, nor too generic, thus becoming too complex to adapt to different contexts.

We want to create a language that allows to represent sequences, of actions and events, and to assign each of them to the subjects that stand in the sequence. For this purpose, (1) we need to understand, collect and analyze, all the concepts and ideas that we want to describe with the IL, in the domain of stories, videos, sequences. (2) We also need to find an existing idea or language, from which to draw inspiration in terms of ideas and concepts, for the creation of the IL elements, that permit to abstract all the concepts of the domain.

For (1) we took inspiration mainly from the case of study of our app and Scratch itself, and from the main concepts of theater script. For (2) After some research, we decided to took inspiration from the standard BPMN, so we will use some concepts and their graphical representation, for naming the elements of the translation in the intermediate language and given an example of their possible graphical representation.

3.3 Language Components

In this Section we want to introduce the components of the Intermediate Language useful for representing the project (that is the subject of the translation), and we suggest a graphical representation similar to BPMN of each of them.

The components are divided in "categories" and we describe each of them and provide their graphical representation[13], the implementation of the Intermediate Language with its component is described in Section 3.5.

In Appendix 1, there is a summary table of the various components, with the suggested graphic representation, the name and a brief description.

SWIMLANE

Swimlanes are rectangular boxes that represent the stage, the participants and the sounds of the project. There are two type of Swimlanes: Pool and Lane. See Figure 3.2

Pool: container for the main roles of the project (what we want to translate). There will be three different types of pools:

- **Stage**: the stage, as well as, the background, middle ground and foreground, of the project. The stage will be the "manager" of the project, the one responsible for the beginning and the end of the flow.
- **Sound**: all the sounds of the project not related to a specific actor (background music or soundtrack, narration, sound effects…)
- **Actors**: the participants of the project, animated (e.g., actors performing actions) or not (e.g., static props).

Lane: the content of the Pool, it contains the flow of the projects. Stage and Sound Pools have Stage and Sound lanes respectively, while Actors Pool has a lane for each participant.

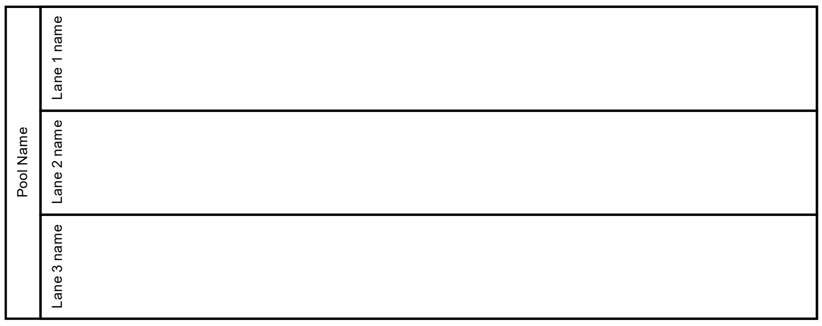


Figure 3.2 – Pool and Lanes

[13] The graphical representation of the intermediate language was generated using the tool available at https://bpmn.io/

EVENT

Events are something that happen/occurs during the flow. They are shown as circles and, in some cases, there are icons within the circles to represent the type of the event. something that occurs during the process. See Figure 3.3

Start Event: the event that indicate the start of the project. There will be only one event of this type and it must be placed in the pool *Stage*.

End Event: the event that indicate the end of the project. There will be only one event of this type and it must be placed in the pool *Stage*.

Middle end Event: an intermediate end event that will be used by the other pools (which are not stage). Once all flow of all the lanes have reached this event, the pool will send a message to the Stage (that, as we said, is a kind of manager of the project) informing it that everyone has finished.

Timer Event: an event that indicate how many seconds the flow needs to wait before the next activity.

Send message Event: an event that send a message to another lane.

Message received Event: an event that wait to receive a message from a "send message" or from a pool. It could have the role of a start event for the flow of a lane, if the message received is the starting message from the Stage, or the role of an intermediate event if the message received is not the starting message.

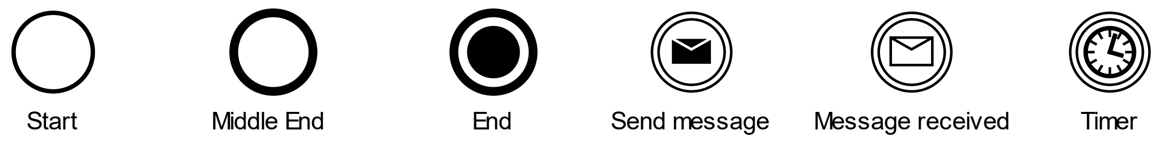


Figure 3.3 - Types of Event

ACTIVITY

Activities are something performed by the participant during the flow. They are shown as rounded rectangle, with names describing the type of task to perform. See Figure 3.4

Image setter activity: activity to set all the initial attributes of the *Stage* or of an *Actor*.

Change costume activity: the activity to change the *Stage* or *Actor* look.

Move: the activity that define the movements of the *Stage* or of an *Actor*, there are different type of move activity:

- **Hide activity:** the activity to make the subject no more visible.
- **Show activity:** the activity to make the subject visible.
- **Glide to x, y activity:** the activity to make the subject glide to a chosen point P (x, y).
- **Move to x, y activity:** the activity to make the subject move to a chosen point P (x, y).
- **Size activity:** the activity to make the subject bigger or smaller (closer or further the viewer)
- **Turn activity:** the activity to make the subject rotate.

Sound: the activity that define a sound of the *Sound* or of an *Actor*. There are two different types of sound action:

- **Text sound activity**: the activity to represent a sound with related text.
- **Generic sound activity**: the activity to represent a sound.

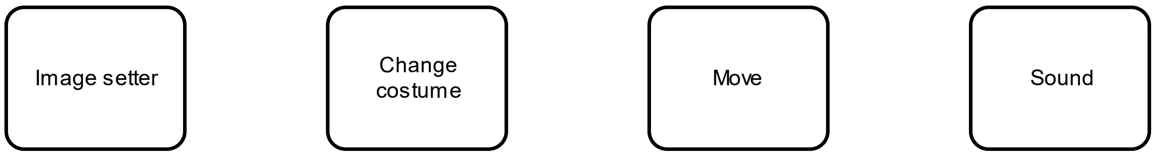


Figure 3.4 - Types of Activity

GATEWAY

The gateway fork it is used to divide the flow in three different parallel flow, all outgoing flows must be executed at the same time. They are shown as diamond shapes with a plus in the middle. See Figure 3.5

When the different flows of the gateway finish all their activity/event, the flow go out from the gateway join and return to a single flow. For simplicity of expression, we have defined the gateway with only three flows and the reason for this choice is explained in the next lines.

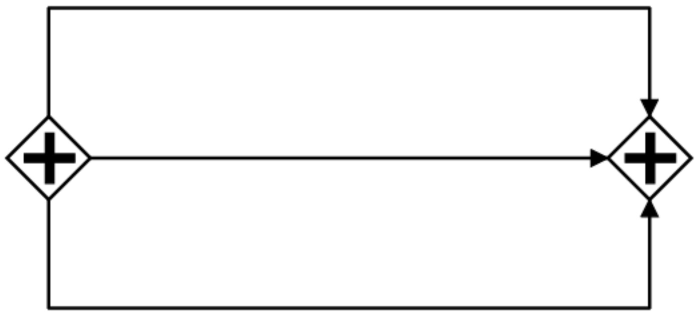


Figure 3.5 - Gateway

The gateway allows to represent activities that occur simultaneously, and we usually need to execute three different type of activities, for this reason we define the gateway with three flow.

For the stage, the three flow can be used for background, middle ground and foreground respectively; for the actors, the first flow can be used for the movement activities, the second flow for the sound activities, and the third flow for the appearance activities; for the sound, the three flow can be used for background music, narration, sound effects, respectively.

If there is the need of more than three flows, there is the possibility to insert one or more gateways inside the gateway to add more parallel flows (see Figure 3.6). If, for example, we are inside a gateway G1 and we want an actor to glide, while rotating and zooming, simultaneously, we can create a new gateway G2 for this purpose.

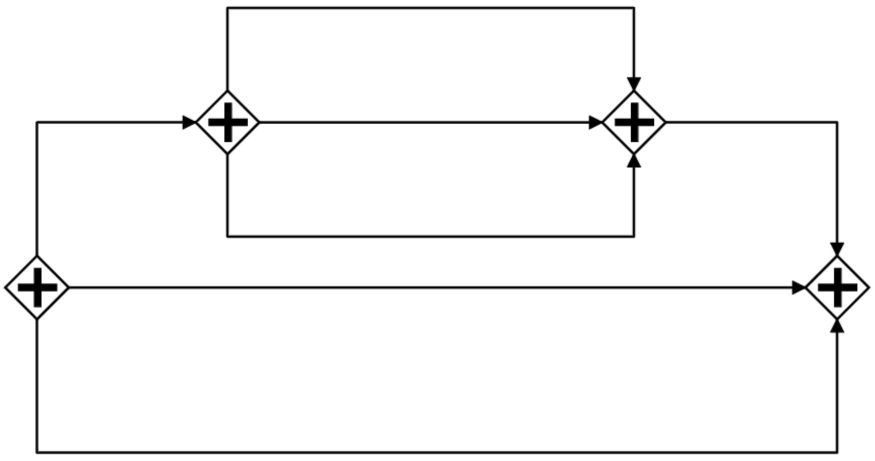


Figure 3.6 - Example of a gateway inside a gateway

FLOW

Flow is useful to show the “order” of the execution of the project, they are shown as line. See Figure 3.7

> **Message Flow**: flow used to show the exchange of messages between one pool and another. A message flow is shown in dotted line with an arrow head.
>
> **Sequence Flow**: is used to connect flow elements. It is shown in solid line with an arrowhead. It shows the order of flow elements. Sequence Flow shows the flow of the execution of events and activities within the same pool.

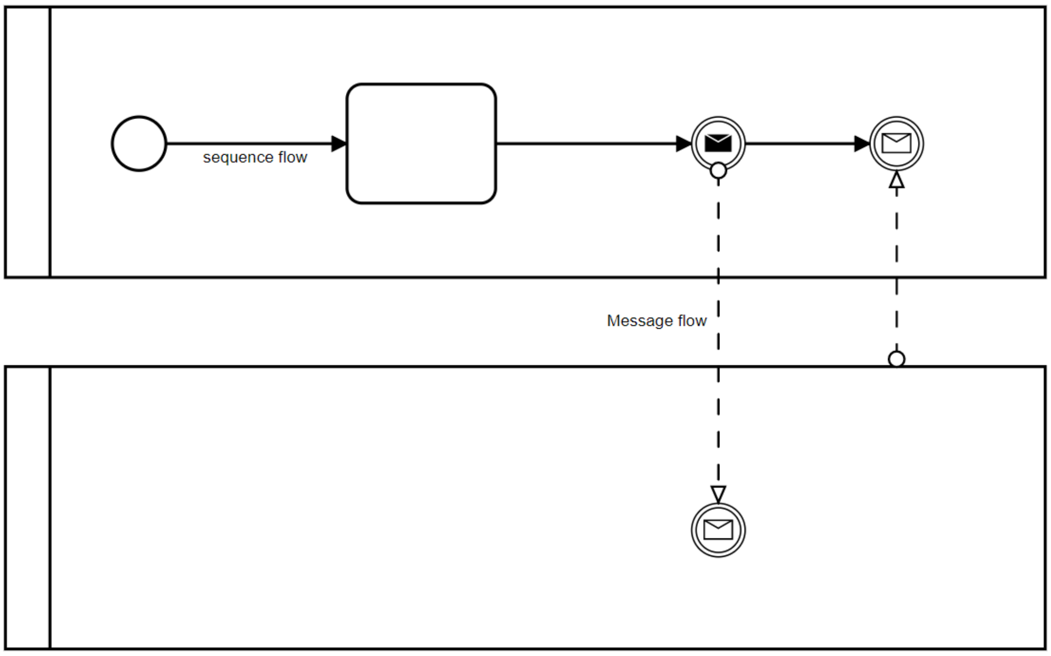


Figure 3.7 - Types of Flow

3.4 Example of components Usage

Here we will analyze and show an example of the usage of the graphical components for describing a sequence/video/animation, showing the interaction between the components and an example of flow from *Start* to *End*. In Figure 3.8 there is the example of an IL project (for a better resolution of the Figure see Appendix 6-A).

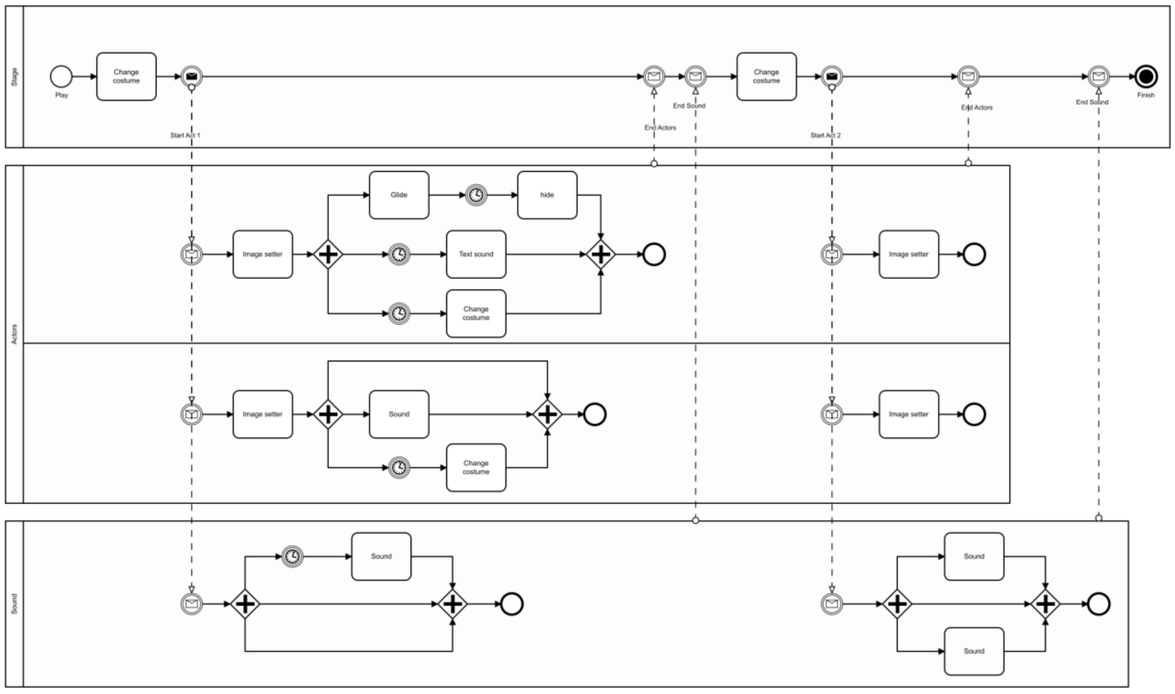


Figure 3.8 - IL example

PROJECT: the project starts and ends, as said, in the stage. It is composed by two main parts that we call "act" taking inspiration from theatre. The act 1 is made of two actors, some actions performed by actors, and a stage sound. The act 2 is made of actors and two different stage sound.

STAGE: in the example the stage is not divided in background, middle ground and foreground, so it is represented by only one of them, for simplicity we say that the stage here is composed only of the background. Before the starting of the two acts there is a *Change costume* activity that change the look of the background. The starting of the act is performed with the *Send message* event and the ending with the two *Message received* events, one from Actors' pool and one from Sound's pool.

ACTORS: in the example there are two actors. The act starts with the *Message received* event from the Stage's pool, and the first activity that we find after it is the *Image setter* activity, which describes the initial state of the actor. Inside the gateway flows there are all the activities performed by the actor and some *Timer* events that manage the time elapsed between them. The first flow is dedicated to the activity of move type, the second flow to the activity of sound type and the third one to the *Change costume* activity.

One or more of the three gateway flow can be empty, as we can see in the example in the second actor. In fact, the first flow of the Act 1, is empty, because the actor does not perform any movement in that act.

When the two actors' flows lane reach the *Middle end* events, the actors' pool sends a message to Stage pool, "reporting" the end of the act for the actors.

SOUND: in the example the sound pool has three main flow (inside the gateway), e.g., one for the background music, one for the sound effects and one for the narration.

As for the actors, when the flow of the sound lane reaches the *Middle end* event, the Sound pool sends a message to the Stage pool "reporting" the end of the act for the sound.

3.5 Data Model Overview

In this Section the data model of the Intermediate Language is presented; to provide a better description, we used some simplified Unified Modeling Language (UML)[14] class diagrams[15].

First, it will be presented a simplified class diagram describing the main contents of a project (Middle Project) in the intermediate language:

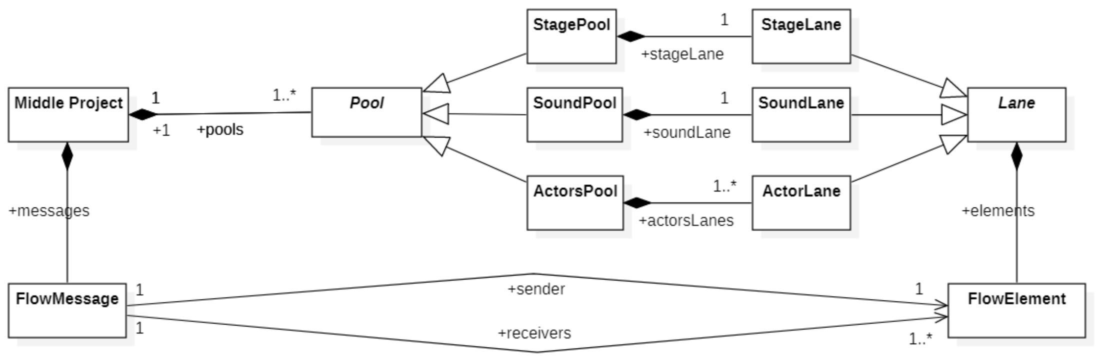


Figure 3.9 - UML Class Diagram of the Intermediate Language

Message flows are represented by the "FlowMessage" class, with the content of the message, the sender, and the receivers. Sequence flows are not represented in the model but are only used graphically to understand the flow and interactions between the various elements; we can see that a sort of sequence flow is represented by the order of the elements in the lists of flow elements.

What is happening during the flow of the project is described by Flow Elements. There are three different type of Flow Elements: Activity, Event, Gateway. In the Figure 3.10 the model of the Flow Elements is showed in a simplified UML Class Diagram.

As shown in the class diagram, the main attributes of the Gateway class are three different ArrayLists of Flow Elements; this is because Gateway divides the flow in three parts, so the three flows from the opening to the closing of the Gateway are

[14] UML is a developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system: https://www.uml.org/
[15] The UML diagrams are generated using the software StarUML available at: https://docs.staruml.io/

contained in it. The reason why there are three flows is described in the Gateway Section in 3.3.

The only elements with a time references are the *Activity* and the *TimerEvent*. Depending to the source language, sometime fields (*startAt, finishAt, duration*), might be missing, so there is a helper class that will help to generate the time references missed. This helper class should be also useful during the translation, to the target language, of the elements that have not the time references in the Intermediate Language (the Event and the Gateway).

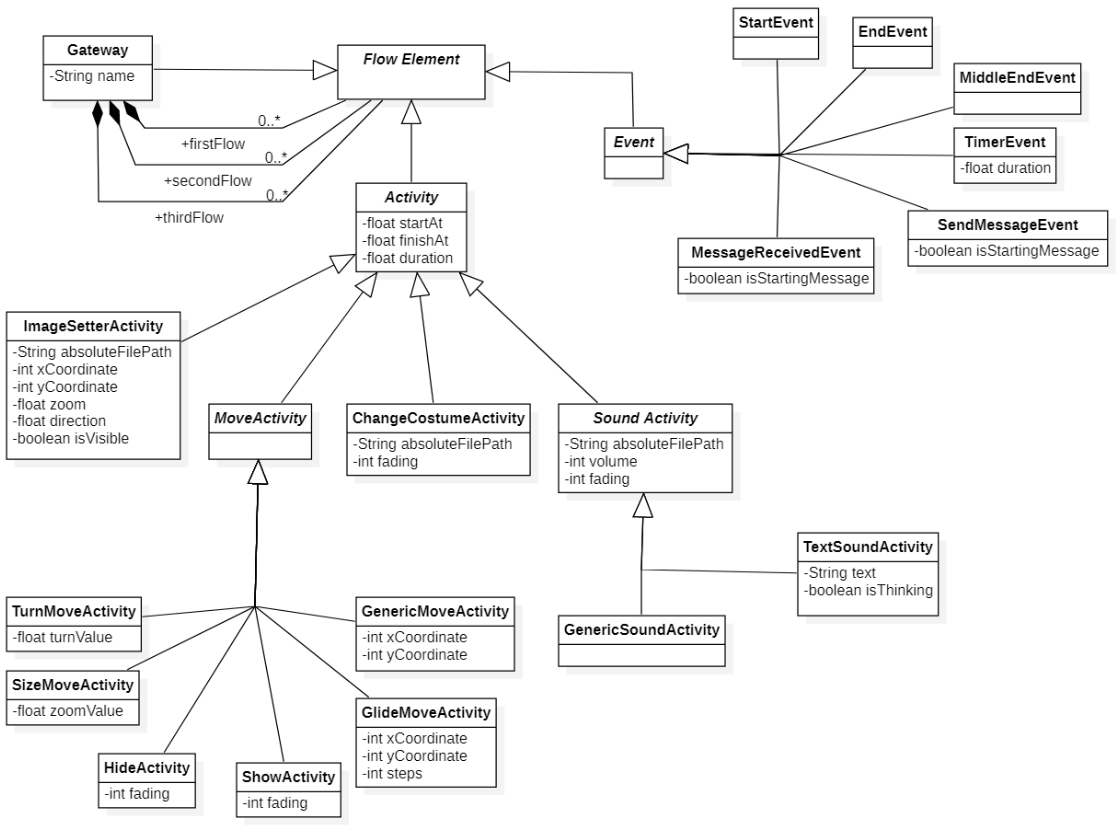


Figure 3.10 - UML Class Diagram describing the Flow Elements

3.6 Abstract Syntax

In this Section the syntax of a core Intermediate Language process is defined, excluding organizational modeling features (i.e., lanes and pools). The definitions are based on (Ouyang, et al., 2007), as previously introduced, and adapted to our case.

Definition 1 (Core IL Process). A core IL process is a tuple $\mathcal{P} = (\mathcal{O}, \mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{T}, \mathcal{S}, \{e^S\}, \mathcal{E}^I, \{e^E\}, \mathcal{E}^{I_M}, \mathcal{E}^{I_T}, \mathcal{G}^F, \mathcal{G}^J, \mathcal{F})$ where:

- $\mathcal{O}$ is a set of objects which can be partitioned into disjoint sets of:
    - activities $\mathcal{A}$,
    - events $\mathcal{E}$,
    - gateways $\mathcal{G}$;
- $\mathcal{A}$ can be partitioned into disjoint sets of tasks $\mathcal{T}$;
- $\mathcal{E}$ can be partitioned into disjoint sets of:
    - start event $\{e^S\}$,
    - intermediate events $\mathcal{E}^I$,
    - end event $\{e^E\}$;
- $\mathcal{E}^I$ can be partitioned into disjoint sets of:
    - intermediate message events $\mathcal{E}^{I_M}$,
    - intermediate timer events $\mathcal{E}^{I_T}$;
- $\mathcal{G}$ can be partitioned into disjoint sets of:
    - parallel fork gateways $\mathcal{G}^F$,
    - parallel join gateways $\mathcal{G}^J$;
- $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$ is the control flow relation, i.e., a set of sequence flows connecting objects.

A core Intermediate language process is a directed graph with nodes (objects) $\mathcal{O}$ and arcs (sequence flows) $\mathcal{F}$. For any node $x \in \mathcal{O}$, input nodes of x are given by $in(x) = \{y \in \mathcal{O} \mid y\mathcal{F}x\}$ and output nodes of x are given by $out(x) = \{y \in \mathcal{O} \mid x\mathcal{F}y\}$. Also, for a core Intermediate language process $\mathcal{P}$, if ambiguity is possible, we use $\mathcal{P}$ as subscripts to each element defined in the tuple $\mathcal{P}$. Next, we define the syntax of a core Intermediate Language model which may comprise a set of core Intermediate Language processes.

Definition 2 (Core IL Model). A core IL model is a tuple $\mathcal{M} = (\mathcal{Q}, \mathcal{F}^M)$ where:

- $\mathcal{Q}$ is a set of core Intermediate Language processes,
- $\mathcal{F}^M \subseteq (\cup_{\mathcal{P}\in\mathcal{Q}} (\mathcal{T}_\mathcal{P} \cup \mathcal{E}^E_\mathcal{P} \cup \mathcal{E}^{I_M}_\mathcal{P}) \times \cup_{\mathcal{P}\in\mathcal{Q}} (\mathcal{T}_\mathcal{P} \cup \mathcal{E}^S_\mathcal{P} \cup \mathcal{E}^{I_M}_\mathcal{P})) \setminus \cup_{\mathcal{P}\in\mathcal{Q}} (\mathcal{O}_\mathcal{P} \times \mathcal{O}_\mathcal{P})$ [16] is the set of message flows between processes.

Definition 1 allows for graphs which are unconnected, not having start or end events, containing objects without any input and output, etc. Therefore, we need to restrict the definition to well-formed processes and models.

Definition 3 (Well-formed core IL Process). A core IL process $\mathcal{P}$ in *Definition 1* is well formed if relation $\mathcal{F}$ satisfies the following requirements:

- $\forall\, s \in \mathcal{E}^S,\ in(s) = \emptyset \wedge |\,out(s)\,| = 1$; i.e., start events have an indegree of zero and an outdegree of one.
- $\forall\, e \in \mathcal{E}^E,\ out(e) = \emptyset \wedge |\,in(e)\,| = 1$; i.e., end events have an outdegree of zero and an indegree of one.
- $\forall\, x \in \mathcal{A} \cup (\mathcal{E}^I \setminus \mathcal{E}^{I_M}{}_S), |\,in(x)\,| = 1 \wedge |\,out(x)\,| = 1$; i.e., activities and non-sender message intermediate events have an indegree of one and an outdegree of one.
- $\forall\, m \in \mathcal{E}^{I_M}{}_S, |\,in(m)\,| = \emptyset \wedge |\,out(m)\,| > 1$; i.e., sender Message intermediate events have an indegree of one and an outdegree of more than one.
- $\forall\, g \in \mathcal{G}^F,\ |\,\mathrm{in}(g)\,| = 1 \wedge |\,out(g)\,| = 3$; i.e., fork gateways have an indegree of one and an outdegree of three.
- $\forall\, g \in \mathcal{G}^J,\ |\,\mathrm{out}(g)\,| = 1 \wedge |\,in(g)\,| = 3$; i.e., join gateways have an outdegree of one and an indegree of three.
- $\forall\, x \in \mathcal{O}, \exists\, s \in \mathcal{E}^S, \exists\, e \in \mathcal{E}^E, s\mathcal{F}^*x \wedge x\mathcal{F}^*e$; [17] i.e., every object is on a path from a start event or to an end event.

Definition 4 (Well-formed core IL Model). A core Intermediate Language model $\mathcal{M}$ given in *Definition 2* is well-formed, iff $\mathcal{Q}$ is a set of well-formed core Intermediate Language processes.

[16] $\cup_{\mathcal{P}\in\mathcal{Q}}$ means the union of the elements in round brackets grouped by each process $\mathcal{P}$ belonging to the set of IL processes $\mathcal{Q}$; "$\setminus \cup_{\mathcal{P}\in\mathcal{Q}} (\mathcal{O}_\mathcal{P} \times \mathcal{O}_\mathcal{P})$ " means to remove, for each $\mathcal{P}$, from the resultant couples all the control flow relations.

[17] $\mathcal{F}^*$ is a reflexive transitive closure of $\mathcal{F}$, i.e. $x\mathcal{F}^*y$ if there is a path from x to y and by definition $x\mathcal{F}^*x$.

3.7 Semantic

In this Section, the semantic of our IL is describes by means of Petri nets, directed bipartite graphs described in Section 2.2.2. Thus, the mapping of well-formed core IL models to Petri nets is established, the mapping is a modified version of the one presented in the research paper (Ouyang, et al., 2008) and (Ouyang, et al., 2007). The reason for choosing Petri nets to describe our semantic has been described in the Section 2.2.

ACTIVITIES, EVENTS and GATEWAYS

Table 3.1 depicts the mapping from a set of IL task, events, and gateways to Petri-net modules. The usage of both labelled and non-labelled transitions is allowed. The labelled transitions model tasks and events. The transitions without a label ("silent" transitions) capture internal actions that cannot be observed by external users. The transition, being labelled with the name of that task or event, models the execution of the task or event.

Table 3.1 is taken from (Ouyang, et al., 2008) where the mapping BPMN object onto Petri-net is analyzed. The main IL components have the same idea and characteristic of BPMN objects.

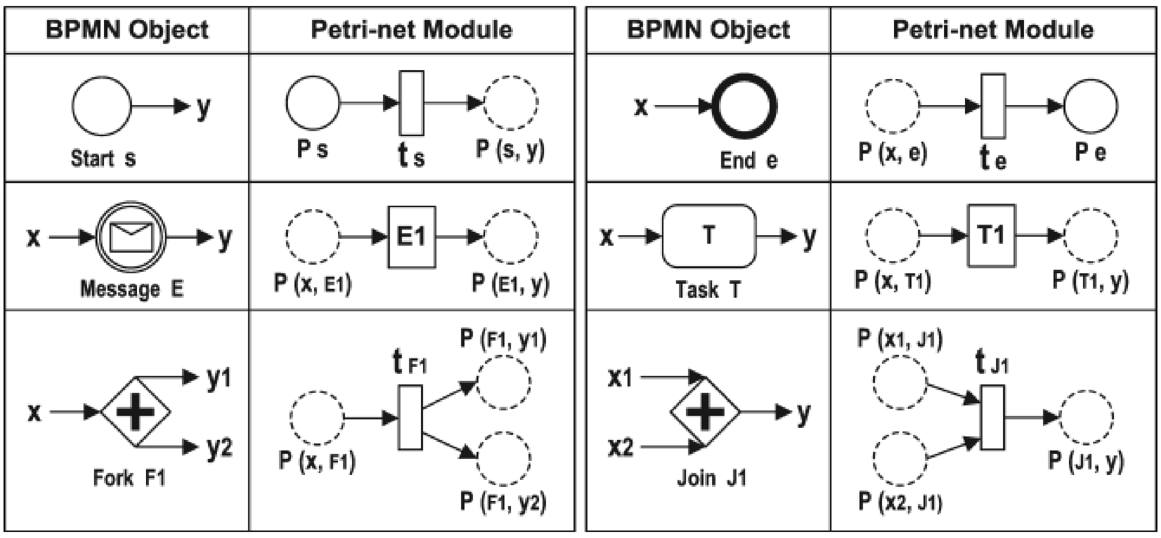


Table 3.1 - Mapping some IL components onto Petri-net modules

Start: the IL start event component is mapped onto a silent transition with an input place and an output place. The silent transition is used to signal when the process starts. A message received component with labeled as starting message is mapped in the similar way.

End: the IL end event and middle end event components are mapped onto a silent transition with an input place and output place. The silent transition is used to signal when the process ends.

Message event: the IL send message event and message received components are mapped onto a transition E, with an input place and an output place.

Task: the IL activity component is mapped onto a transition T, with an input place and an output place.

Gateway fork: the IL gateway component fork is mapped onto a silent transition with an input place and three output places.

Gateway join: the IL gateway component join is mapped onto a silent transition with three input places and one output place.

MESSAGE FLOW

The next Figures will show the flow message from a send message event or a pool to a message received event. A message flow is used to show the transmission of messages between different pools or lanes.

Message from send message event to message received event: the transition that represents sending the message has a flow from itself to the place that precedes the transition modeling the received message. In the Figure 3.11 there is the mapping from a send message to a message received event that has the role of a start event, different mapping is applicable to a message received event that has the role of an intermediate event as it is shown in the Figure 3.12.

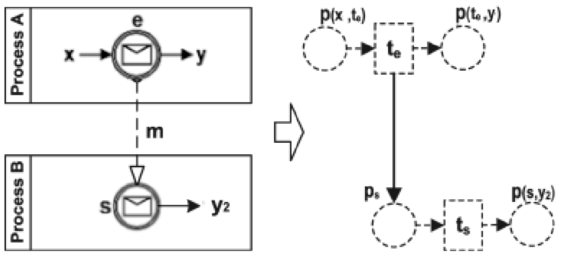


Figure 3.11 - Message from Send message event to Message received event

Message from Pool to message received event: more complex mapping is for the message that represent the message of ending of the “work” from the actors to the stage and it is represented in the Figure 3.12. In an abstract way, it can be mapped to a transition with one or more incoming arc from the places modeling the middle end event and an outgoing arc to the transition modeling the received message event. As we can see, the message received event is mapping differently in the two case according to the role it has (start event or intermediate event).

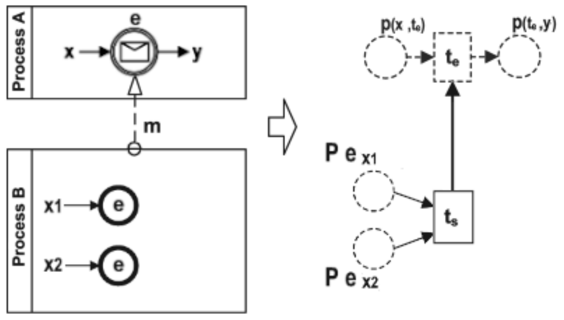


Figure 3.12 - Message from Pool to Message received event

INITIAL MARKING CONFIGURATION

The initial state of an IL model can be specified by the initial marking of the corresponding Petri net model, as mentioned before, a start event signals the start of the IL process. For this reason, the start place for the event Start is the only place being marked (with a token drawn as a big black dot) in the initial marking of the corresponding Petri net model.

When the token will arrive in the end place for the event End, we will be sure that all the process is ended.

4 Case Study

The second part of this thesis focuses on the development of a desktop application for storytelling, that allows the user to create a story, choosing characters and playing their roles with their voice, also offering the possibility to watch the whole story through the Scratch web site or App and share it with parents or friends. The application and the translation from the application language to the Scratch language, will be an example of use of the Intermediate Language.

In the next Sections the case study will be analyzed and explained, first presenting the background and then describing the application in detail.

4.1 Background: The LYV Project

An important inspiration for this thesis is to be credited to Lend Your Voice (LYV) project[18] from ARCSLab, which has been funded by the Polisocial Award[19] 2016.

LYV is a software for the development of interactive stories, with a dual purpose: support in learning English for young people with specific learning disorder and vocal re-education for children and young people with intellectual disabilities or difficulty in speech.

LYV, built on Electron platform, translated a story in the *sb2* Scratch format, and, with the help of ScratchX, it permitted to have interactions with the story (to record and to analyse voice, checking and visualizing the prosody) during the execution of the story itself.

The graphical structure of the Application Prototype Scene Editor, showed in the Section 5.3.4, and the main idea of describing Stories in the graphical way with Scene's timeline composed by blocks, is highly inspirated from LYV editor. The idea to translate the story in a Scratch file is also taken from LYV project, where the story was translated in sb2. Since January 2019, there is a new version of Scratch: sb3; the two output formats (sb2 and sb3) are strictly different and sb2 is based on Flash (no longer supported), we decided to translate in the new version of Scratch because the previous version is working correctly only in the Scratch 2.0 Dekstop.

[18]http://www.polisocial.polimi.it/wp-content/uploads/2016/07/46_LYV_Sbattella_no_VIDEO_finanziato.pdf
[19] http://www.polisocial.polimi.it/en/home-en/

4.2 Description of the application

After using LYV, the Esagramma[1] association realized the usefulness of similar software to be used in multimedia laboratories and, more generally, to be used in the association's activities.

For this reason, we decided to create an application similar to LYV that is simpler and less powerful but easier to use and that permit to create more personalized stories, that would allow to create simple stories, deciding how to compose the story, the looks of backgrounds, characters and objects, and interpreting the main characters.

The idea behind the translation from the graphical representation of the story described in the application, to the story in a Scratch executable file, is showed in the Figure 4.1. The story in the source language is translated in a story in the intermediate language, then the story in the intermediate language is translated in a story in the target language. In the next Sections, we describe more in details the story in the three different languages and the translation process.

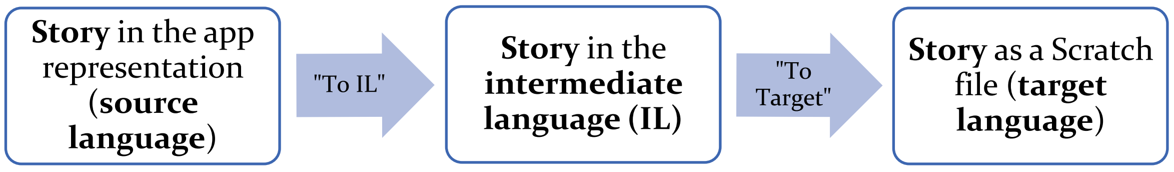

Story in the app representation (source language)
"To IL"
Story in the intermediate language (IL)
"To Target"
Story as a Scratch file (target language)

Figure 4.1 - Steps of the translation: case study

4.2.1 Target

The main targets are some subjects from Esagramma[1] and from Amicinsieme[2] with whom the application tests will be done. In general, these are people with intellectual disabilities and can be both young and adult.

We can identify two types of app users, for which we expect the following benefits:

Educators / Volunteers: an app with which to create animated stories in a simple way, allowing the users of the association to participate, telling a story together and interpreting the protagonists (or one of the protagonists). The possibility of reviewing what is going on during the creation of the scenes, almost in real time, listening to the voice within the evolution of the scene. The aim that we would like to achieve is to improve the communication and social skills of the users, also making them interact with each other (i.e., it is possible to improve communication also with other users by working together as protagonists of the same story).

Users: an app that allows to decide backgrounds, characters, and objects, that will participate in the story and what will happen to them, describing the movements, the looks and the sounds/voices of them. The possibility to interpret the protagonists, lending the voice (dubbing the character) identifying with them. The aim is having fun and obtaining the complete story as a result that they can review and listen to again, even sharing it with third parties (family, friends ...).

4.2.2 Functional requirements

The educator (together with the users or not) must be able to:

Create a new story: the educator must be able to create a new story

Use an existing story: it must be possible to use one or more base stories, created in advance and made available in the app, so that there is, from the first use of the app, an example ready to be modified or to be used to directly record the voices of the children.

Create the .sb3 file of the story: it must be possible to create a sb3 file that contains the complete story in order to be able to play and share it with Scratch.

Edit a saved story: It must be possible to edit and customize one of the saved stories. Therefore, both before and after the eventual creation of the sb3 file, it must be possible to save the story in the app, so that it can be modified at a later time.

The details expected for the creation of a story are the following:

- **Add a title**
- **Add a short description + details**
- **Edit / create scenes**:
  - Choose the background of the scene
  - Choose characters present in the scene
  - Choose objects present in the scene
  - Get characters in and out
  - Make the characters / objects in the scene move / rotate
  - Let the characters speak through voice recording / file upload + written dialogues
  - Choose the moods of the characters in the various moments of the scene

- Choose the appearance of the characters and objects in the various moments of the scene
- Set background music through voice recording / file upload
- Insert narration through voice recording / file upload
- Insert voice recording / file upload sound effects

The users must be able to:

Insert custom characters: deciding the characteristics of the character (the same goes for backgrounds and objects).

- Choose the name of the character
- Enter some details about the character
- Insert the default image of the character
- Insert the images of the emotion of the character
- Insert other images of the character

Play a character: which can be a specific character or several characters; to interpret it he must be able to record his own voice in the dialogues of the scenes.

Register sound: they can create story sounds with object or musical instruments, register them and adding them to the scene as sound effect, background music or sound of the characters/objects of the scene.

Review the story with their own voice inside: being able to review the entire story in sb3 format via Scratch online or Scratch desktop.

Share the created story: being able to show the entire story to others via Scratch web site.

4.3 The source language

The case study for this thesis is inspired from LYV project. In particular, the subject is the stories creation and translation into Scratch file. The stories are created with an application prototype that will be presented in the Section 4 and Section 6.1.1. A story is a list of scenes, the scenes are described by a list of blocks and initial settings. In the next sessions Application Blocks and Scratch Blocks are presented.

4.3.1 A Story in the App

The story, in the application representation, is a list of scenes. Each scene has a background, the two lists of participants (characters and objects), the list of initial settings one for each participant, and the list of blocks representing the actions and sounds happening in the scene (more details in the next Section). Blocks are the main part of the scene representation and their order is based on the "starting time" of the block grouped by subject (the subject can one of the participants or one of the music types related to the stage).

The full data model of the App elements is presented in the Section 6.1.1.

4.3.2 Application Blocks

A scene can be described with the blocks here presented: the actions and appearance of the scene participants and the different type of sound and music played during the scene.

Table 4.1 - List of Application blocks for scene description

Block Name	Applicable to	Features
Enter on stage	• Characters • Objects	The entry on the stage for a specific participant. The user can decide where the subject will arrive after the entry and the duration of this action. The user needs to also choose the type of entry (from where the subject will enter), or if it will enter directly in the place, without gliding.
Leave the stage	• Characters • Objects	To leave the stage for a specific participant. The user can decide the duration of the action and the type of that: if the subject will glide to an edge of the screen or if it will disappear on the spot.

Move	• Characters • Objects	To move the subject in the screen. The user can decide the duration of the movement and the destination point coordinates.
Rotate	• Characters • Objects	The rotation of a subject on the spot. The user can decide the rotation duration and the quantity (expressed in degrees)
Zoom	• Characters • Objects	The zoom in or out of a subject on the spot. The user can decide the zoom duration and the quantity (expressed in degrees)
Speak	• Characters	The speech of a character. The user can register an audio or choose one from the directory and insert the text of the speech. The duration of this block depends on the audio length.
Sound	• Characters • Objects • Narrator • Background music • Sound effects	The sound of the stage or of a participant. The user can register an audio or choose one from the directory. The duration of this block depends on the audio length.
Emotion	• Characters	The emotion of the character. The user can choose the duration and the emotion from the emotions available (the look of the character will change if there is an image for the emotion chosen)
Change costume	• Characters • Objects	The appearance of the participant. The user can choose the duration and one of the personalized images previously inserted in the participant.

The "applicable to" column is useful to understand for each block to which scene component can be applicable. The main "components" of the scene are what we call participants: scene characters and scene objects; the music of the scene is described by three "components": narrator, background and sound effects.

Some remarks regarding some blocks are described below.

In the resulting scene, for all the duration of the *emotion* block or the *change costume* block, the character or object will change appearance depending on the emotion image or personalized image chosen, but when there is none of these blocks the appearance will depend on the default image chosen for the scene character or the scene object.

The *speak* block needs the text, because in the resulting scene, every speaker will present the bubble with the written speech, and it is the only way to recognize the speaker. In the future we would like to add some animations during the speaking time and adding the possibility to choose if the bubble is shown or not.

Describing the content of a scene, in addition to the blocks, there is also the possibility to choose the initial settings of the participants: position (x and y coordinates), visibility, rotation and zoom.

4.4 The target language

In this Section we will present a general description of a story in Scratch.

4.4.1 A Story in Scratch

A Scratch file is a ZIP archive, with the extension .sb3, which contains information encoded in a JSON file and project media in separate files.[20]

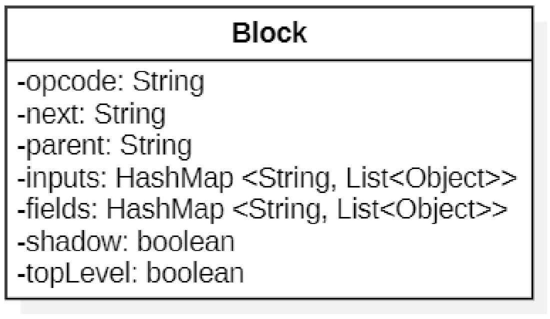
In our case the project media are the images for backgrounds and actors (called *costumes* in Scratch), and the sounds; the files where they are stored have the names beginning with their MD5 checksums followed by the file extension, as requested by the sb3 file format[21]. The content of the story is described in the *project.json* file.

The data model of the Scrath project is presented in the Section 6.1.2 . Briefly, the story in Scratch model is a Project containing a list of Targets, where Target is the stage or a participant of the story (called *sprite*). The Stage is always the manager of the story, and it will contain the music and manage all the scene changes. Each Target (i.e., the stage and the sprites) contains all the information about the action and sound performed by the subject in a list of blocks.

4.4.2 Scratch Blocks

In the first part of this Section, we will introduce the "block" concept in the Scratch file format with some examples. Then, in the end of this Section and in the next ones, we will talk about "Scratch blocks" as the graphical representation that the block has in the Scratch editor, without considering the on-disk internal format.

In the Figure 4.2 , the block in the Scratch file format is showed from the point of view of the Scratch data model in our application.



```
Block
-opcode: String
-next: String
-parent: String
-inputs: HashMap <String, List<Object>>
-fields: HashMap <String, List<Object>>
-shadow: boolean
-topLevel: boolean
```

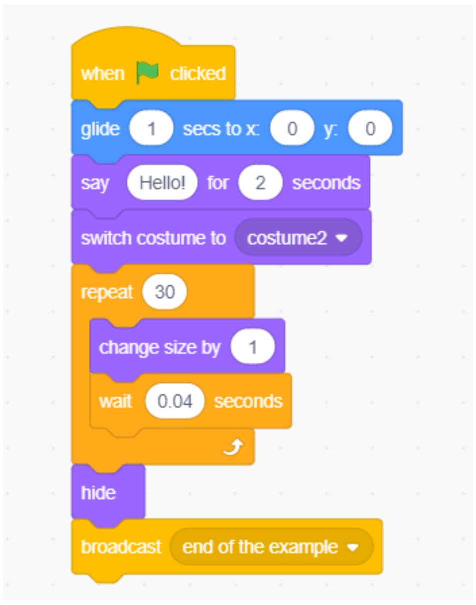
Figure 4.2 - Block class from the Scratch data model of our application

[20] https://en.scratch-wiki.info/wiki/Scratch_File_Format
[21] https://en.scratch-wiki.info/wiki/Scratch_File_Format#Project_Files

The block in the Scratch file format has the following properties:

- **opcode**: It is a string with the name of the block.
- **next**: The ID of the next block.
- **parent**: The ID of the preceding block.
- **inputs**: An object associating names with arrays representing inputs.
- **fields**: An object associating names with arrays representing fields.
- **shadows**: A boolean value, True if it is a shadow block and false otherwise.
- **topLevel**: A boolean value, False if the block has a parent and true otherwise.

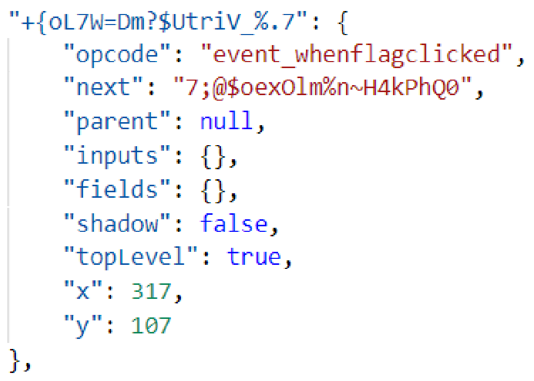
It is easier to describe the attributes of the block with an example. In the Figure 4.3, there is an example of some blocks from the Scratch editor, and we will analyze all of them, one by one, from the top to the bottom.



```
when [flag] clicked
glide (1) secs to x: (0) y: (0)
say (Hello!) for (2) seconds
switch costume to (costume2 ▾)
repeat (30)
    change size by (1)
    wait (0.04) seconds
hide
broadcast (end of the example ▾)
```

Figure 4.3 - Scratch editor view of the example

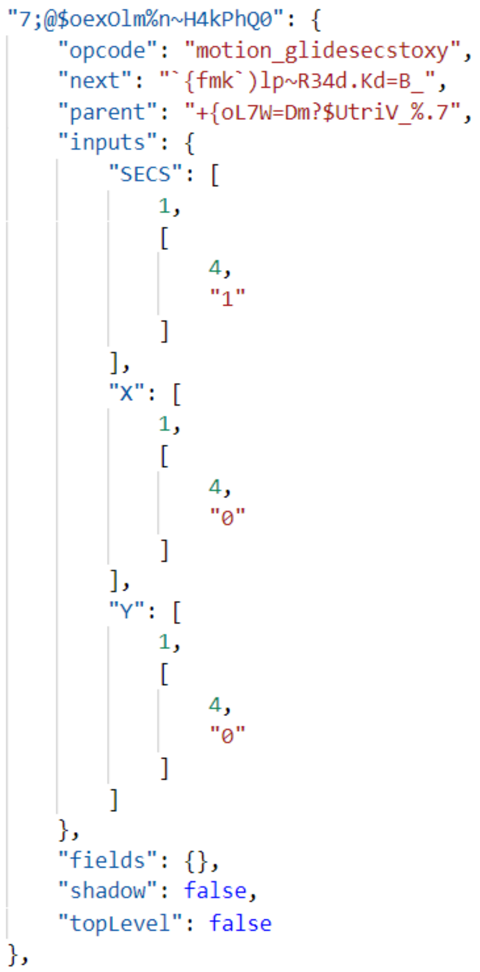
In Figure 4.4, we can see the block “When flag clicked” and we can notice that: 1) *parent* is *null* because it is the first block, for the same reason *topLevel* is True; 2) *inputs* and *fields* are empty; 3) *shadow* is false because it is not a shadow block; 4) *x* and *y* are the coordinates of the position of the first block of the stack; the last two properties are not present in the Figure 4.2 because Scratch automatically sets them at the default value.



```
"+{oL7W=Dm?$UtriV_%.7": {
    "opcode": "event_whenflagclicked",
    "next": "7;@$oexOlm%n~H4kPhQ0",
    "parent": null,
    "inputs": {},
    "fields": {},
    "shadow": false,
    "topLevel": true,
    "x": 317,
    "y": 107
},
```

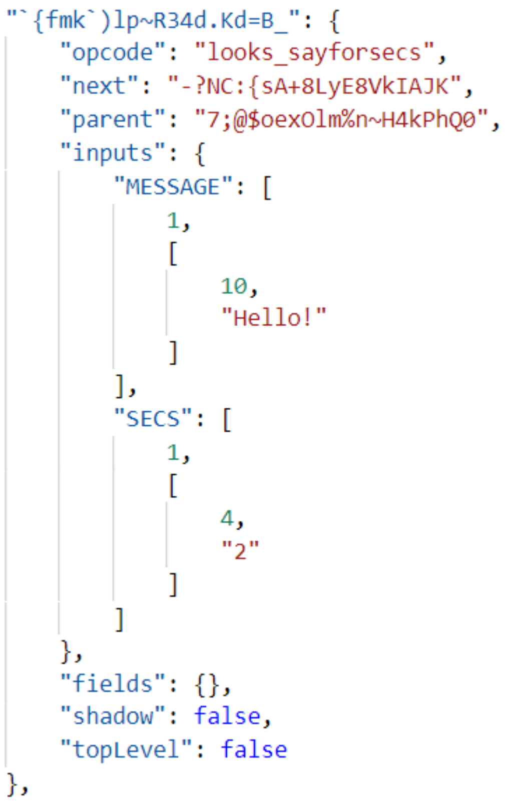
Figure 4.4 - When flag clicked

In Figure 4.5 and Figure 4.6, we can see two examples of the *inputs* property, that contain the information inserted by the user: duration of the glide, final position after the glide (in the first one) and the message to say and its duration (for the second one).



```
"7;@$oexOlm%n~H4kPhQ0": {
    "opcode": "motion_glidesecstoxy",
    "next": "`{fmk`)lp~R34d.Kd=B_",
    "parent": "+{oL7W=Dm?$UtriV_%.7",
    "inputs": {
        "SECS": [
            1,
            [
                4,
                "1"
            ]
        ],
        "X": [
            1,
            [
                4,
                "0"
            ]
        ],
        "Y": [
            1,
            [
                4,
                "0"
            ]
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

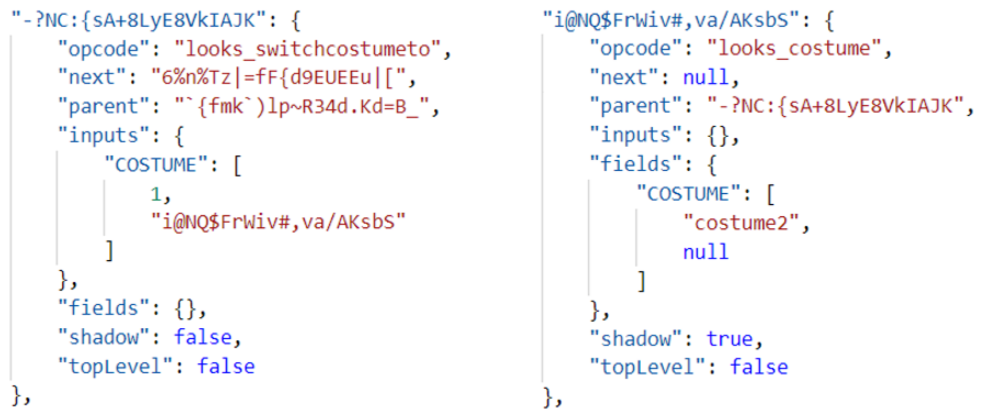
Figure 4.5 – Block: Glide 1 secs to x=0, y=0



```
"`{fmk`)lp~R34d.Kd=B_": {
    "opcode": "looks_sayforsecs",
    "next": "-?NC:{sA+8LyE8VkIAJK",
    "parent": "7;@$oexOlm%n~H4kPhQ0",
    "inputs": {
        "MESSAGE": [
            1,
            [
                10,
                "Hello!"
            ]
        ],
        "SECS": [
            1,
            [
                4,
                "2"
            ]
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

Figure 4.6 – Block: Say "Hello!" for 2 seconds

An example of a shadow block is showed in the Figure 4.7. We can see that the block *Switch costume* (in the left side of the Figure) has, as an input, a shadow block (in the right side of the picture). The shadow block contains all the information about the costume chosen as an input of the block in the property *fields*.

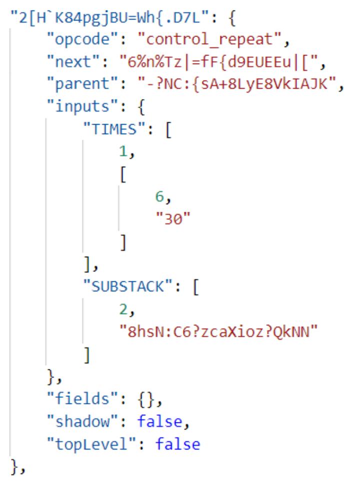


```
"-?NC:{sA+8LyE8VkIAJK": {
    "opcode": "looks_switchcostumeto",
    "next": "6%n%Tz|=fF{d9EUEEu|[",
    "parent": "`{fmk`)lp~R34d.Kd=B_",
    "inputs": {
        "COSTUME": [
            1,
            "i@NQ$FrWiv#,va/AKsbS"
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

```
"i@NQ$FrWiv#,va/AKsbS": {
    "opcode": "looks_costume",
    "next": null,
    "parent": "-?NC:{sA+8LyE8VkIAJK",
    "inputs": {},
    "fields": {
        "COSTUME": [
            "costume2",
            null
        ]
    },
    "shadow": true,
    "topLevel": false
},
```

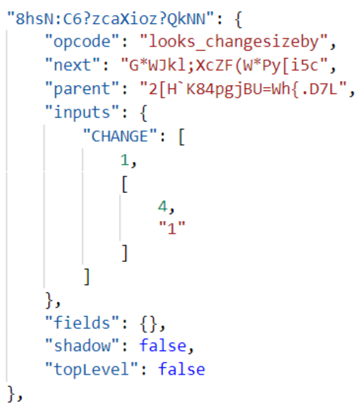
Figure 4.7 - Block: Switch costume to "costume2"

In the next three Figures there are the tree blocks of the loop. The Figure 4.8 shows the *Repeat* block, where we can notice that in the inputs fields there are two elements, one that says how many loops to do, and the other that represents the *substack* (that are the other two blocks of the Figure 4.9 and Figure 4.10).



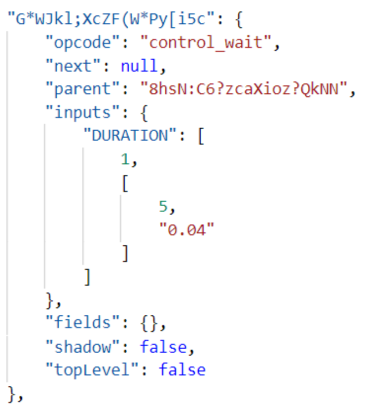
```
"2[H`K84pgjBU=Wh{.D7L": {
    "opcode": "control_repeat",
    "next": "6%n%Tz|=fF{d9EUEEu|[",
    "parent": "-?NC:{sA+8LyE8VkIAJK",
    "inputs": {
        "TIMES": [
            1,
            [
                6,
                "30"
            ]
        ],
        "SUBSTACK": [
            2,
            "8hsN:C6?zcaXioz?QkNN"
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

Figure 4.8 - Block: Repeat 30 times



```
"8hsN:C6?zcaXioz?QkNN": {
    "opcode": "looks_changesizeby",
    "next": "G*WJkl;XcZF(W*Py[i5c",
    "parent": "2[H`K84pgjBU=Wh{.D7L",
    "inputs": {
        "CHANGE": [
            1,
            [
                4,
                "1"
            ]
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

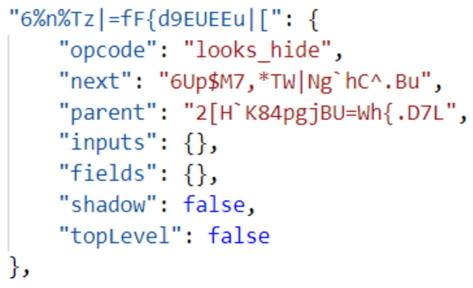
Figure 4.9 - Block: Change size by 1



```
"G*WJkl;XcZF(W*Py[i5c": {
    "opcode": "control_wait",
    "next": null,
    "parent": "8hsN:C6?zcaXioz?QkNN",
    "inputs": {
        "DURATION": [
            1,
            [
                5,
                "0.04"
            ]
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

Figure 4.10 - Block: Wait 0.04 seconds

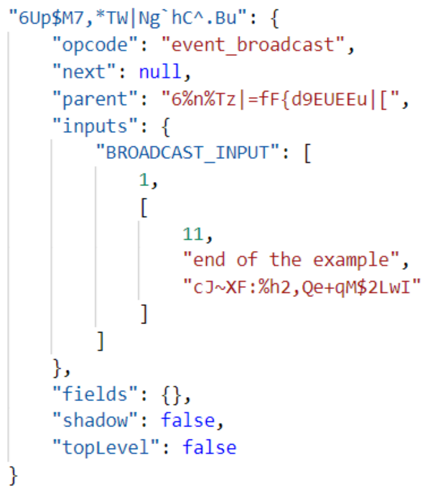
As shown in Figure 4.11, the *hide* block has nothing special to point out.



```
"6%n%Tz|=fF{d9EUEEu|[": {
    "opcode": "looks_hide",
    "next": "6Up$M7,*TW|Ng`hC^.Bu",
    "parent": "2[H`K84pgjBU=Wh{.D7L",
    "inputs": {},
    "fields": {},
    "shadow": false,
    "topLevel": false
},
```

Figure 4.11 – Block: hide

The Figure 4.12 shows the last block of the stack. The *inputs* property of the *Broadcast* block contains the name and the ID of the message to broadcast.



```
"6Up$M7,*TW|Ng`hC^.Bu": {
    "opcode": "event_broadcast",
    "next": null,
    "parent": "6%n%Tz|=fF{d9EUEEu|[",
    "inputs": {
        "BROADCAST_INPUT": [
            1,
            [
                11,
                "end of the example",
                "cJ~XF:%h2,Qe+qM$2LwI"
            ]
        ]
    },
    "fields": {},
    "shadow": false,
    "topLevel": false
}
```

Figure 4.12 - Block: Broadcast "end of the example"

In the Table 4.2 we describe the Scratch blocks that we use in our case study.

Table 4.2 - List of relevant Scratch blocks for case study

Block Name	**Features**
Go to x, y	To choose x coordinate and y coordinate, moving the subject from the current point to the new point.
Glide *n* secs to x,y	The same of "Go to x, y" but with glide in *n* seconds.
Turn right/left *n* degrees	To turn the subject of *n* degrees in the right of left direction.
Point in direction n	To choose *n* to change the direction of the subject.
Say *text* for *n* seconds	To choose text and the duration of it, the chosen text will appear for *n* seconds in a speech bubble on the top corner of the subject.
Think *text* for *n* seconds	The same for "say" but with a thought bubble.
Switch costume / backdrops to *name*	To change costume (sprite) or backdrops (stage) in *name* one.
Change size by *n*	To increase or decrease the size of the subject depending by the *n* chosen.
Set size to *n* %	To set the size to *n* % dimension (100% is default)
Show / hide	To show or hide the subject.
Play sound *name* until done	To start a sound by *name* and wait until its finish.
When flag clicked	To start the process when the green flag is clicked.

When I receive *message*	To receive a *message*.
Broadcast *message*	To send a *message* to the other (sprites and/or stage)
Wait *n* seconds	To wait *n* seconds.
Repeat *n* times	To repeat *n* times the content. In this block you can insert other blocks and it will repeat them *n* times.

4.4.3 Mapping App blocks onto Scratch blocks

In this Section, we will analyze which Scratch blocks are useful for replicate, in the final animation, the behavior desired by the app blocks. Therefore, we will try to map the app blocks onto Scratch blocks. This mapping was useful for the translation from the story in the IL to the story in a Scratch file (showed in Section 4.5.3). The next formulas present: on the left side the application blocks described in Section 4.3.2 and on the right side the Scratch blocks described in Section 4.4.2.

Enter on stage ⟹ `go to` *`x1, y1`* `+ show + glide to` *`x2,y2`* `in` *`n`* `seconds` . (Where x1 and y1 are the coordinates of the point where the element will start the entrance and x2 and y2 are the coordinates of the point of the final position of the subject)

Leave the stage ⟹ `glide to` *`x, y`* `in` *`n`* `seconds + hide` . (Where x and y are the coordinates of the exit point)

Move ⟹ `glide to` *`x1, y1`* `in` *`n`* `seconds` . (Where x1 and y1 are the coordinates of the point and n the duration of the movement)

Rotate ⟹ `turn right` *`n2`* `degrees` . (Where *n* is the rotation size)

Zoom ⟹ `change size by` *`n`* . (Where *n* is the zoom size)

Speak ⟹ `start sound` *`name`* `+ say` *`text`* `for` *`n`* `seconds` . (where *name* is the name of the sound that we want to play, and *n* is the duration of this sound)

Sound ⟹ `play sound` *`name`* `until done` . (where *name* is the name of the sound that we want to play)

Emotion/ Change costume ⟹ `switch costume to` *`name1`* (Where *name1* is the emotion image or the personalized image)

4.5 Description of the translation

In this Section the translation of a Story from the application representation to the Scratch file is presented, this translation is the main idea with which we created and tested the Intermediate Language. As previously said, we can divide the translation in two parts: “from source language to IL” and “from IL to target language”.

4.5.1 A Story in the Intermediate Language

The story in the intermediate language is a project that contains pools that are list of lanes, following the idea previously presented (Stage, Actors, Sound).

The project starts with the start event and finish with the end event, the manager of the process is the Stage that contains the two events. Other than the start and end of the story, the Stage pool manages the changing of the backgrounds and the activities related to the start and the end of the scenes.

All the other lanes (Actors and Sound) contain the main actions happening during the story. The starting of the scenes is represented by a *message received* event, which receives the message from the Stage when the new scene is starting, and the ending of the scene is represented by the *middle end* event.

4.5.2 From source language to IL

With the following “formulas” we want to briefly describe how a story is translated from the source language to the IL. On the left side of the formulas there are the application blocks described in Section 4.3.2 and some concept related to the application (for instance, the “starting of the story”), on the right side of the formulas there are the IL components described in Section 3.3.

Starting of the story ⟹ `Start event in Stage lane`

Ending of the story ⟹ `End event in Stage lane`

Starting of a scene:

(Stage lane) ⟹ `Change costume Activity with duration 0 seconds in Stage lane + Send message event in Stage lane + Timer Event with duration m seconds in Stage lanes` (Where m is the duration of the scene)

(Sound lane) ⟹ `Message Received event in sound lane + Gateway in sound lane`

(Actors lanes) ⟹ `Message Received events in actors lanes + Image setter activities in actors lanes + Gateways in actors lanes`

`FLOW MESSAGE`: A flow message element is created for every "starting of a scene". The element has the new send message event as "sender" and a list of all the new received events as "receivers".

The content of the scene:

The gateways created in actors' lanes during the "*starting of a scene*", will be populated like that: the first flow will contain all the Move Activities of the corresponding actor, the second flow all its Sound Activities and the third flow all its Change costume Activities.

The gateway created in sound lane during the "*starting of a scene*", will be populated like that: the first flow will contain the narrator Sound Activities, the second flow the background music Sound Activities and the third flow all the sound effects Sound Activities.

SCENE BLOCKS. In this Section we analyze in detail the translation from the Scene blocks of the Application representation to the Story in the IL. They will be inserted in the gateways as above explained.

N.B.: In all the following: "duration n seconds" , n is the duration of the block.

Enter on stage (with glide) ⟹ Move to x1, y1 Activity with duration 0 seconds + Show Activity with duration 0 seconds + Glide to x2,y2 Activity with duration n seconds . (Where x1 and y1 are the coordinates of the entering position chosen, x2 and y2 are the coordinates of the final position of the subject)

Enter on stage (without glide) ⟹ Move to x1, y1 Activity with duration 0 seconds + Show Activity with duration 0 seconds + Timer Event with duration n seconds . (Where x1 and y1 are the coordinates of the final position of the subject)

Leave the stage (with glide) ⟹ Glide to x, y with duration n seconds + hide with duration 0 seconds . (Where x and y are the coordinates of the exit point)

Leave the stage (without glide) ⟹ Hide with duration 0 seconds + Timer Event with duration n seconds

Move ⟹ Glide to x, y Activity with duration n seconds . (Where x and y are the coordinates of the final position of the subject).

Zoom ⟹ Size Activity with duration n seconds

Rotate ⟹ Turn Activity with duration n seconds

Speak ⟹ Text sound Activity with duration n seconds

Sound ⟹ Generic sound Activity with duration n seconds

Emotion ⟹ Change costume Activity with duration 0 seconds + Timer Event with duration n seconds

Change costume ⟹ Change costume Activity with duration 0 seconds + Timer Event with duration n seconds

4.5.3 From IL to target language

In this Section we briefly describe how a story is translated from the Intermediate Language to the target language, with the following "formulas". On the left side of the formulas there are the IL components described in Section 3.3, on the right side of the formulas there are the Scratch blocks described in Section 4.4.2.

Events:

Start Event ⟹ `When flag clicked`

Timer Event ⟹ `Wait` *`n`* `seconds` (Where *n* is the timer duration).

Send message Event ⟹ `Broadcast "message" in the current stack`

Message received Event (Actor lane) ⟹ `When I receive "message" in the current sprite`

Message received Event (Sound lane) ⟹ `When I receive "message" in the Stage`

Gateway:

(Actor lane) ⟹ `Broadcast "message" + 3x (When I receive "message") in the actor sprite`

(Sound lane) ⟹ `Broadcast "message" + 3x (When I receive "message") in the stage`

The gateway adds a *Broadcast message* block to the current stuck and three new *When I receive message* block, one for every flow. The flow elements of the gateway will be stacked in the new stacks.

Activities:

Image setter Activity ⟹ `switch costume to` *`name1`* `+ wait` *`n`* `seconds + switch costume to` *`name2`* . (Where *name1* is the image name, *n* is the changing duration, and *name2* is the default

Change costume Activity ⟹ `switch costume to` *`name1`* `+ wait` *`n`* `seconds + switch costume to` *`name2`* . (Where *name1* is the image name, *n* is the changing duration, and *name2* is the default image)

Hide Activity ⟹ `hide`

Show Activity ⟹ `show`

Glide to x, y Activity ⟹ `glide` *`n`* `seconds to` *`x1, y1`* . (Where x1 and y1 are the coordinates corresponding to final position of the subject after the move)

Move to x, y Activity ⟹ `go to` *`x1, y1`* . (Where x1 and y1 are the coordinates corresponding to final position of the subject after the move)

Text sound Activity ⟹ `start sound` *`name`* `+ say` *`text`* `for` *`n`* `seconds` . (Where *name* is the name of the sound that we want to play, and *n* is the duration of this sound)

Generic sound Activity ⟹ `play sound` *`name`* `until done` . (Where *name* is the name of the sound that we want to play)

Zoom Activity ⟹ `Repeat` *`a`* `times (change size by` *`b`* `+ wait` *`c`* `seconds )`

Rotate Activity ⟹ `Repeat` *`a`* `times (turn right` *`b`* `degrees + wait` *`c`* `seconds )`

The “Zoom” and “Rotate” formulas need more explanation:

In Scratch does not exist the idea of change size or turn in a given amount of time, so, to obtain this result, we need to loop different blocks of Scratch, as seen in the formulas above. Introducing the loop in the Scratch program, however, might generate lagging problems.

To address this, first of all, an analysis of the lags in Scratch programs was done; we discovered that the execution of one Scratch block generates a waste of time that we analyzed to be 17/20 %. With the use of single or a couple of blocks at a time, the lag is not visible; for this reason, the blocks that are not inside a loop should not cause any problems.

We consider a problem only the lag generated by a loop (that we want to use in zoom and rotation cases), because we can have tens or hundreds of blocks executed together and this can generate visible lags.

Therefore, for the following formulas, we made some approximation based on empirical data and on what we have observed from the results in Scratch execution.

In the formulas of zoom and rotate activities, *a* is calculated in the following way:

$$a = t \times f$$

Where t is the duration in seconds of the rotation/zooming and f represents the number of frames per seconds.

$$b = q \div a$$

Where q is the "quantity" of rotation/zoom (q = *how much we want to rotate/zoom in total?*)

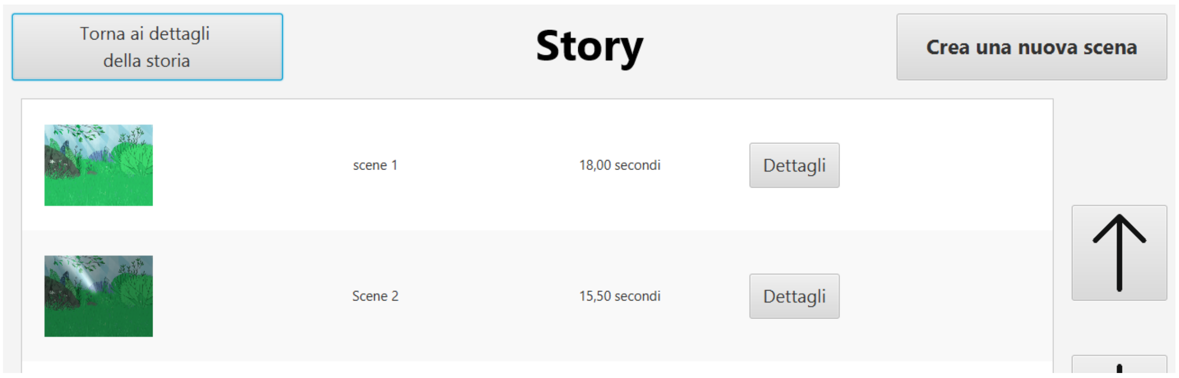
$$c = 0{,}04 \ \ and \ \ f = 15$$

The number of times to repeat the loop *(f)* is chosen like that because 15 frames per seconds are enough for the sensibility of the human eye (to see the movement continuous and fluid). The value of the waiting time *c* is the delay in seconds to avoid too many frames in a second (with the chosen value they are at maximum 25).

4.6 Translation example

In this Section we present a practical example of the translation from the source language to the target language. For better understanding and for simplicity, we use the graphical representation of the source language and the target language. As for the IL components, they are represented with the proposed, BPMN-based, graphical representation that, as previously discussed, is intended only to ease human understanding; at the moment, it is not possible to directly define the IL components from a graphical point of view (see Section 3.5 for the data model of the IL).

Application. The example is a simple story with just two small scenes: scene 1 and 2 (Figure 4.13).



	scene 1	18,00 secondi	Dettagli
	Scene 2	15,50 secondi	Dettagli

Figure 4.13 – Example: scenes of the story APP

Intermediate Language. In the Figure 4.14 (for a better resolution of the Figure see Appendix 6-B), the full story in the Intermediate Language is showed. In the next paragraphs we will describe it more in detail, starting from the Stage pool and the Sound pool, then analyzing the Actors in scene 1 and in scene 2.

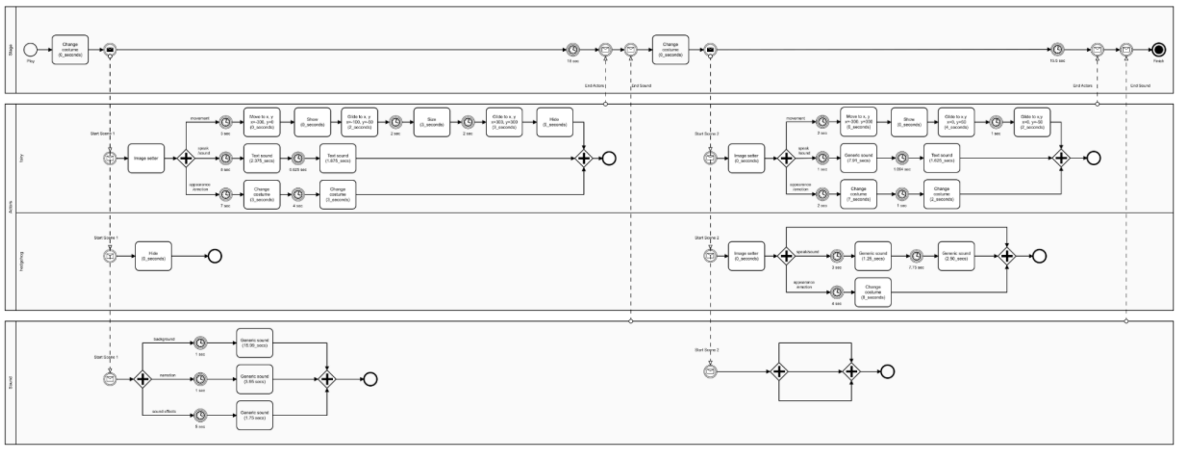


Figure 4.14 - Example: Story in the IL

In the Figure 4.15, we can see that the Stage pool for the scene 1 has only one activity: the "change costume" activity that set the background image at the starting of the scene 1. After the activity there is the "Send message" event that send to all the other lanes the message "Start Scene 1". The duration of the scene is represented by the event "timer". At the end of the scene there are the two "Message received" events that receive the message "End Actors" and "End Sound". As showed in the Figure 4.14, the same is for the scene 2.

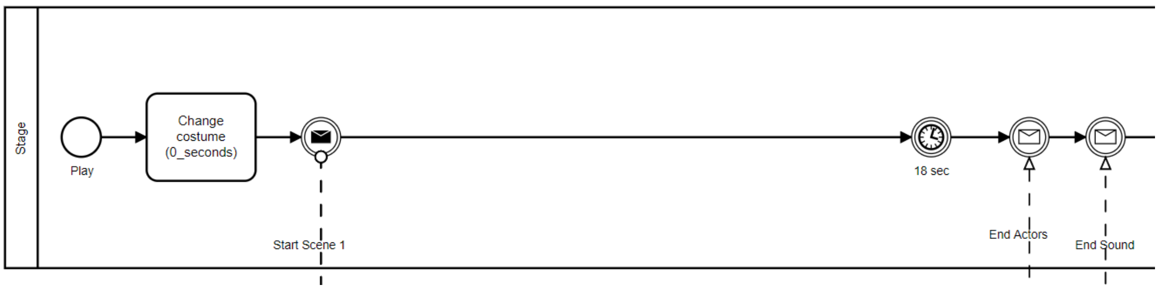


Figure 4.15 - Example: Stage scene 1 IL

As showed in Figure 4.21, the first scene is created inside the application with one *sound* block for the narration, one for the background music and one for the sound effects. The three sound blocks are translated in the IL as showed in the Figure 4.16: at the start of the scene the pool "sound" receives the message "Start Scene 1" from the Stage, then a "Gateway" divides the flow in three different flows: background, narration and sound effects. Inside each of them there are the corresponding block for the scene 1, in our case there is one activity each, preceded by events that indicate how long to wait before starting the activity.

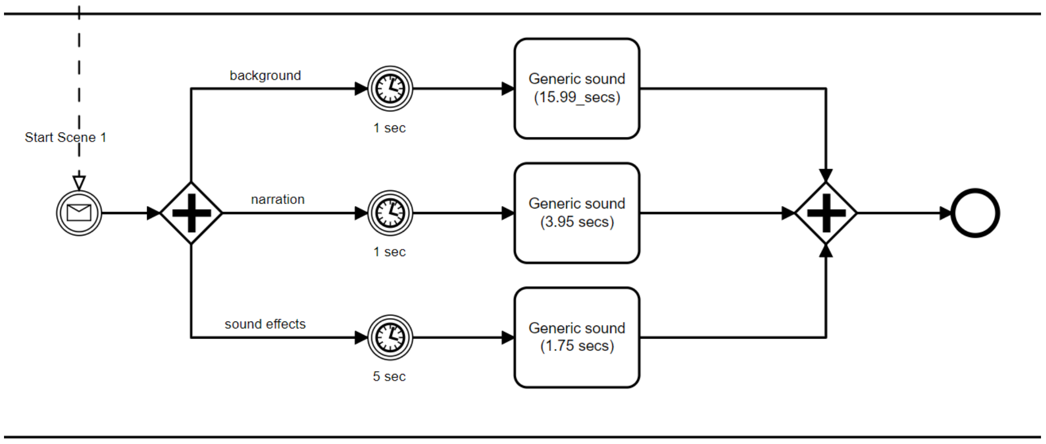

Start Scene 1
background
1 sec
Generic sound (15.99_secs)
narration
1 sec
Generic sound (3.95 secs)
sound effects
5 sec
Generic sound (1.75 secs)

Figure 4.16 - Example: Sound scene 1 IL

In the scene 2, instead, there are any music blocks (neither narration nor background music nor sound effects), this in the IL is translated as showed in the Figure 4.17. Up to the “Gateway” element, everything is the same of the scene 1, but inside the gateway flows there are no elements.

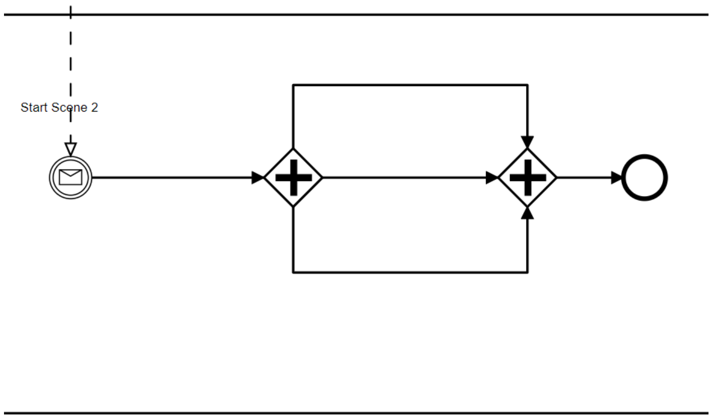
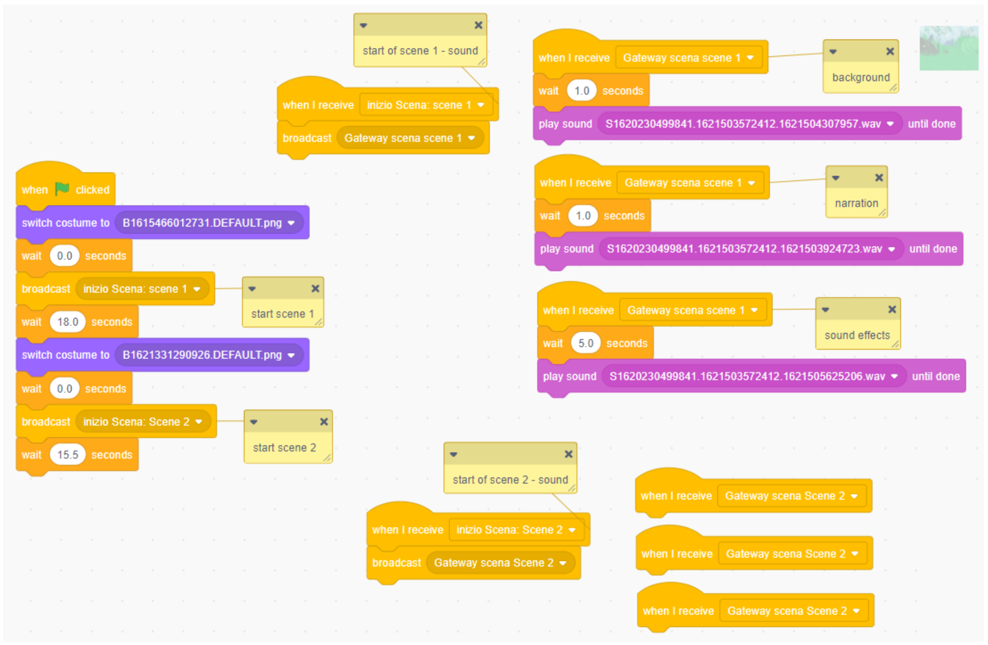


Figure 4.17 - Example: Sound scene 2 IL

Scratch. In the Figure 4.18, the story from the Stage point of view of Scratch blocks is showed. In particular, in the left side, the stack that start with the block “when flag clicked” is the translation of the Stage pool showed in the . The other stacks of blocks are the translation of the sound pool described in the previous paragraphs. Now we will describe them more in details.

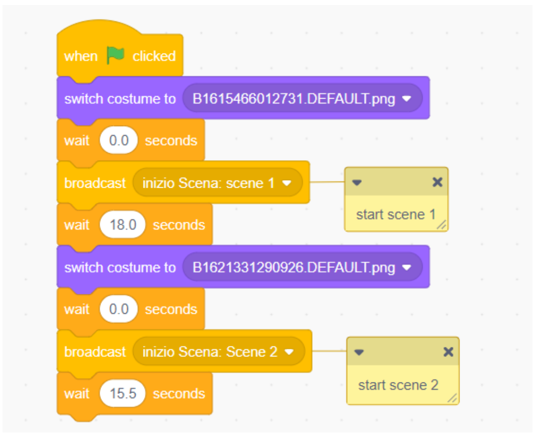


```
when [flag] clicked
switch costume to B1615466012731.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: scene 1        // start scene 1
wait 18.0 seconds
switch costume to B1621331290926.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: Scene 2        // start scene 2
wait 15.5 seconds

when I receive inizio Scena: scene 1   // start of scene 1 - sound
broadcast Gateway scena scene 1

when I receive Gateway scena scene 1   // background
wait 1.0 seconds
play sound S1620230499841.1621503572412.1621504307957.wav until done

when I receive Gateway scena scene 1   // narration
wait 1.0 seconds
play sound S1620230499841.1621503572412.1621503924723.wav until done

when I receive Gateway scena scene 1   // sound effects
wait 5.0 seconds
play sound S1620230499841.1621503572412.1621505625206.wav until done

when I receive inizio Scena: Scene 2   // start of scene 2 - sound
broadcast Gateway scena Scene 2

when I receive Gateway scena Scene 2

when I receive Gateway scena Scene 2

when I receive Gateway scena Scene 2
```

Figure 4.18 - Example: Stage SCRATCH

Stage pool. In the Figure 4.15, we can see the detail of the scene 1 in the Stage pool, and in the Figure 4.19 its translation in Scratch blocks (with also the scene 2). The

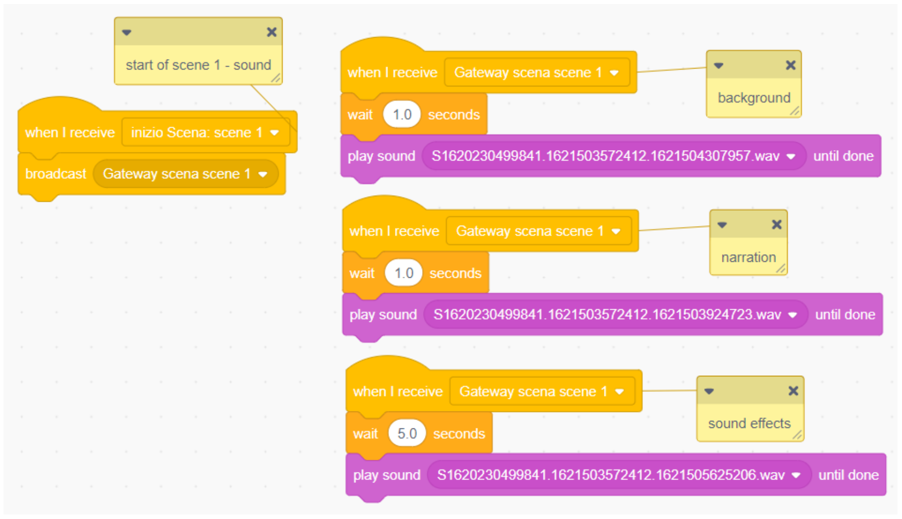
Start event is translated in the “when flag clicked” , which is the first block activated by Scratch when the user click on the flag that will play the animations. The *change costume* activity is translated in a “change costume” block and a “wait” block of 0 seconds (because the activity change costume has duration of 0 seconds). The translation of *Send message* is the block “broadcast”. When Scratch execute the “broadcast *message_1*” block, automatically send the “*message_1*” to the Stage and all the sprites, thus all the “when I receive *message_1*” will be executed. The event *timer* is translated in the Scratch block “wait” and it is the duration of the scene. The two *Message received* events are not translated because they are not useful for the Scratch story execution. The scene 2 is translated in the same way of the scene 1.



```
when clicked
switch costume to B1615466012731.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: scene 1        start scene 1
wait 18.0 seconds
switch costume to B1621331290926.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: Scene 2        start scene 2
wait 15.5 seconds
```

Figure 4.19 - Example: Stage scene 1 SCRATCH

Sound pool. In the Figure 4.16, we can see the detail of the scene 1 in the Sound pool and in the Figure 4.20, its translation to the Scratch blocks inside the *Scratch Stage* is showed. The block commented as “start of scene 1 – sound” is the translation of the *Message received* event. The first block after the “when I receive *inizio Scena: scene 1*” is the “broadcast” that is the translation of the *Gateway fork* and it sends the message of the gateway to the other three stacks: one for the background, one for the narration and one for the sound effects. Thus, for each flow there is the corresponding Scratch stack. The *Timer* events are translated as “wait” blocks and the *Generic sound* activities as the “play sound until done” blocks. For the scene 2 is the same but the three stacks corresponding to the three flows are empty for the same reason previously explained.



```
start of scene 1 - sound

when I receive inzio Scena: scene 1
broadcast Gateway scena scene 1

when I receive Gateway scena scene 1      background
wait 1.0 seconds
play sound S1620230499841.1621503572412.1621504307957.wav until done

when I receive Gateway scena scene 1      narration
wait 1.0 seconds
play sound S1620230499841.1621503572412.1621503924723.wav until done

when I receive Gateway scena scene 1      sound effects
wait 5.0 seconds
play sound S1620230499841.1621503572412.1621505625206.wav until done
```

Figure 4.20 - Example: Stage-Sound scene 1 SCRATCH

Scene 1

Application. In the application (see Figure 4.21), the first scene has only one character and there is only one block for the narration, one block for the background music and one block for the effect sounds. The character “Fairy” (“*Fatina dei fiori*” in the Figure) has: three yellow blocks *action* (enter, change size and exit), two cyan blocks *speak*, one red block *emotion* (“Angry”) and one light red block *appearance* (“Flying”).

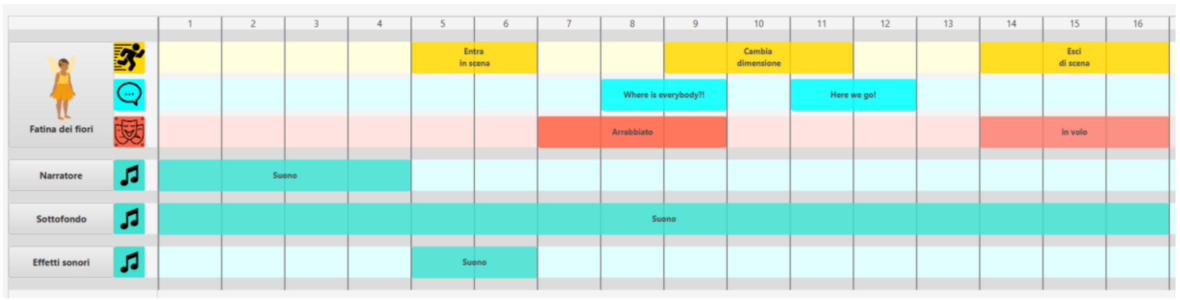


Figure 4.21 - Example: scene 1 APP

Intermediate Language. In the Figure 4.22, the scene 1 from the “Fairy” point of view is shown. The first *Activity*, after the reception of the message “start scene 1”, is the *Image setter* that set all the initial information about the character in the scene like the position, the rotation and the size (in this case we did not modify the settings so they will be the default ones). After the *Gateway*, the three flows are managed as previously said: the movement flow, the speak/sound flow and the emotion/appearance flow.

Movement flow. We can notice that the yellow block *enter* from the APP is translated in three different *Activities*: "Move to x,y" (x,y is the point from where the character starts its movement to enter), "show"(before this activity the character is not visible) and "Glide" (the activity that perform the entering movement until the character reach the final position). The *Change size* block of the app is translated in a "Size" activity. The *exit* block is translated in two *Activities*: "glide to x,y" (x,y is the point of the edge where the character finish is movement and "exit") and "hide"(after the exit the character is no more visible in the scene).

Speak/sound flow. In the Figure 4.22 we can see the two "Text sound" activities corresponding to the two *Speak* blocks of the application.

Appearance/emotion flow. The *Emotion* block and the *Appearance* block are translated as two "Change costume" activities.

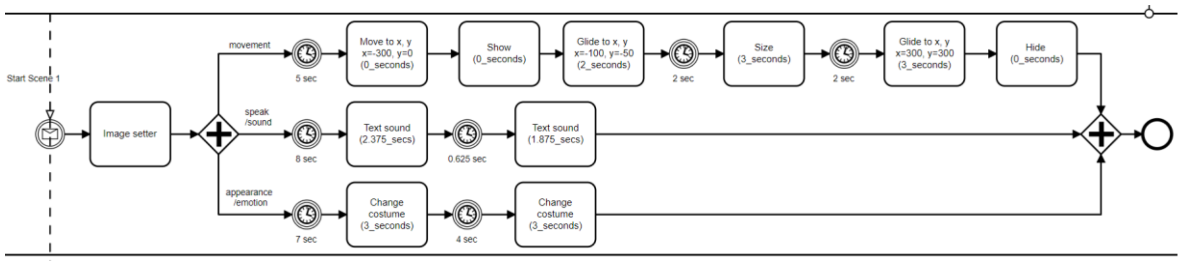
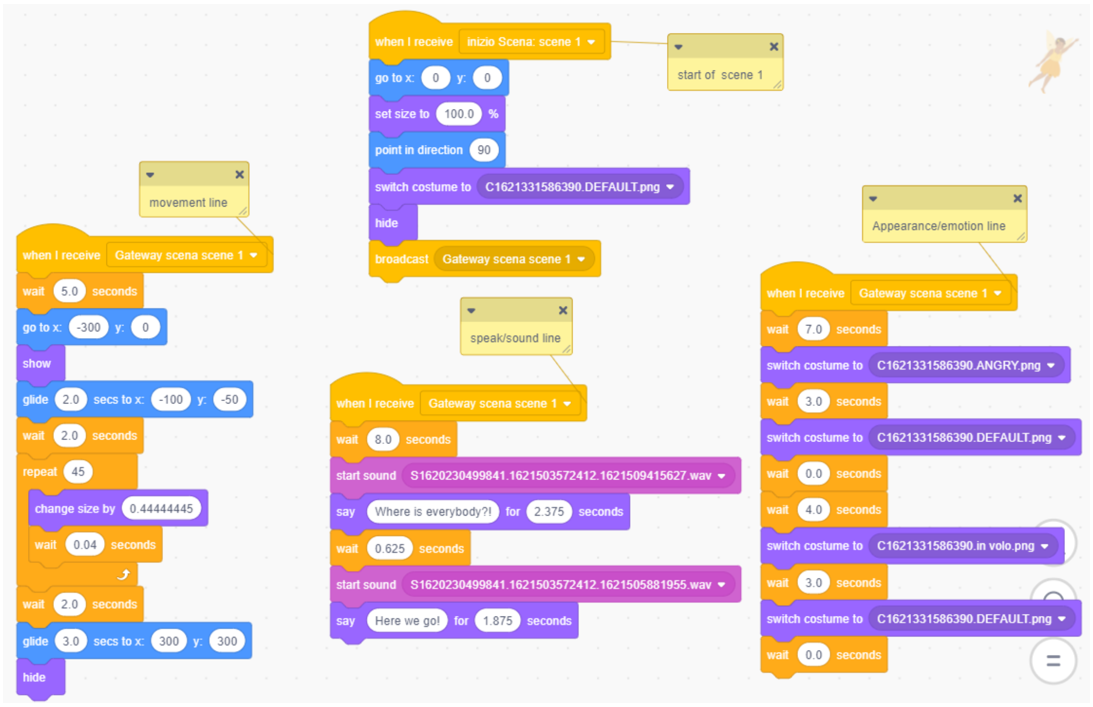


Figure 4.22 - Example: Fairy scene 1 IL

Scratch. In the Figure 4.23, the result of translation in Scratch for scene 1 for the "fairy" is showed. It is possible to notice that the scene, for the character, starts when the message "start Scene 1" is received with the block "When I receive". After the start of the scene, there are the five blocks corresponding to the *Image setter* activity, and, as the last block of the first stack there is a "broadcast", that sends the message "Gateway scene1" and start three more stack, that are the three flow created by the *gateway* in the IL. Some activities are directly translated to the corresponding Scratch block, while others needed more than one Scratch block, and we will now focus on these last ones.



```
movement line

when I receive [Gateway scena scene 1 ▾]
wait (5.0) seconds
go to x: (-300) y: (0)
show
glide (2.0) secs to x: (-100) y: (-50)
wait (2.0) seconds
repeat (45)
    change size by (0.44444445)
    wait (0.04) seconds
wait (2.0) seconds
glide (3.0) secs to x: (300) y: (300)
hide
```

```
when I receive [inizio Scena: scene 1 ▾]        start of scene 1
go to x: (0) y: (0)
set size to (100.0) %
point in direction (90)
switch costume to [C1621331586390.DEFAULT.png ▾]
hide
broadcast [Gateway scena scene 1 ▾]
```

```
speak/sound line

when I receive [Gateway scena scene 1 ▾]
wait (8.0) seconds
start sound [S1620230499841.1621503572412.1621509415627.wav ▾]
say (Where is everybody?!) for (2.375) seconds
wait (0.625) seconds
start sound [S1620230499841.1621503572412.1621505881955.wav ▾]
say (Here we go!) for (1.875) seconds
```

```
Appearance/emotion line

when I receive [Gateway scena scene 1 ▾]
wait (7.0) seconds
switch costume to [C1621331586390.ANGRY.png ▾]
wait (3.0) seconds
switch costume to [C1621331586390.DEFAULT.png ▾]
wait (0.0) seconds
wait (4.0) seconds
switch costume to [C1621331586390.in volo.png ▾]
wait (3.0) seconds
switch costume to [C1621331586390.DEFAULT.png ▾]
wait (0.0) seconds
```

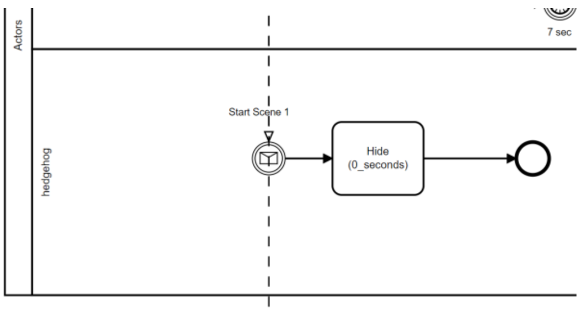
Figure 4.23 - Example: Fairy scene 1 SCRATCH

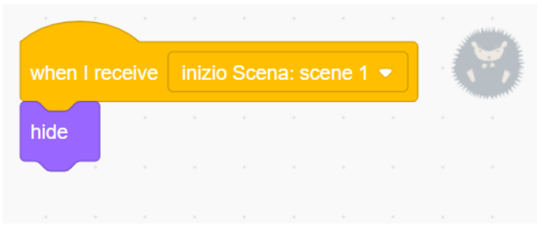
Movement line. We can notice that the *Size* activity from the IL is translated in three different blocks (following the loop formula explained at the end of the previously Section).

Speak/sound flow. We can see that the *Text* sound activity is translated in two Scratch blocks: "start sound" (which performs the recorded file) and "say" (which prints the line on the screen).

Appearance/emotion flow. We notice that every *Change costume* activity is translated in four Scratch blocks: "switch costume" (which has as input the image that we want to show with the activity), "wait" (which is the duration of the activity), "switch costume_default" (which perform the changing of the appearance to come back to the default one, because we want the appearance change only for the duration of the activity and then come back to the default one) and an "empty" wait block (because the default image has not a duration).

As we can notice in the Figure 4.21, in the scene 1, the character "hedgehog" is not a participant, but it is a character of the story because it is a participant in the scene 2. Therefore, in the IL the character has a "hide" activity (see Figure 4.24) and the block "hide" is also the only block for "hedgehog" in the first scene as a translation of the IL (see Figure 4.25).





```
when I receive [inizio Scena: scene 1 ▾]
hide
```

Figure 4.25 - Example: Hedgehog scene 1 SCRATCH

Figure 4.24 - Example: Hedgehog scene 1 IL

Scene 2

Application. In the application (see Figure 4.26), the scene 2 has two participants. The character "Fairy" ("*Fatina dei fiori*" in the Figure) has: two yellow blocks *action* (enter and move), one dark cyan block *sound* and one cyan block *speak*, two light red blocks *appearance*.

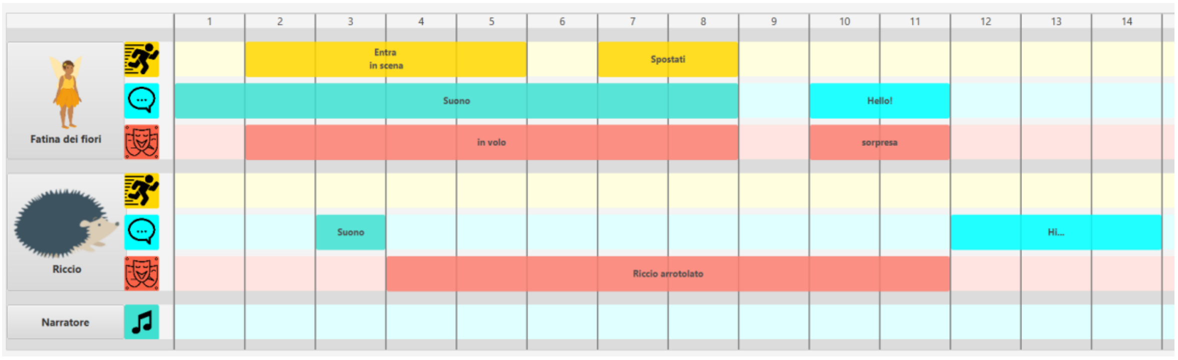


Figure 4.26 - Example: scene 2 APP

Intermediate Language. In the Figure 4.27, the scene 2 for the actor "Fairy" is showed, while in the Figure 4.28, it is showed for the actor "hedgehog". In this last, we can notice that there is one of the three flows inside the *gateway* that is empty, this is because the "hedgehog" do not have any action block in the scene 2.

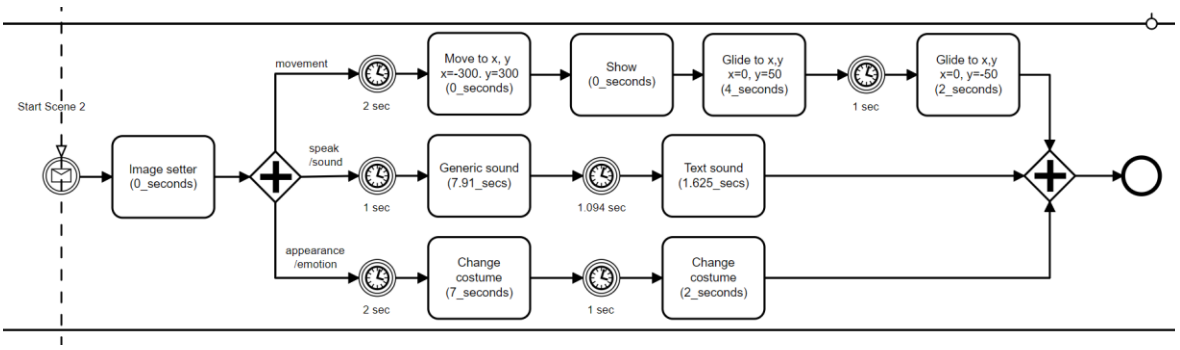


Figure 4.27 - Example: Fairy scene 2 IL

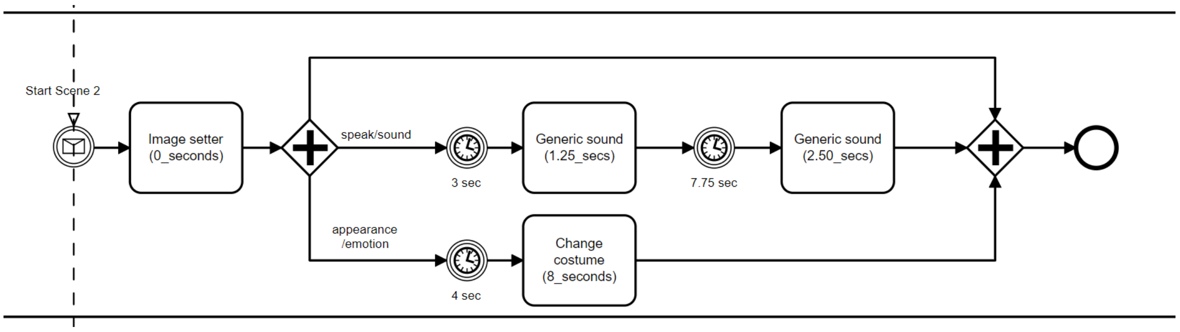
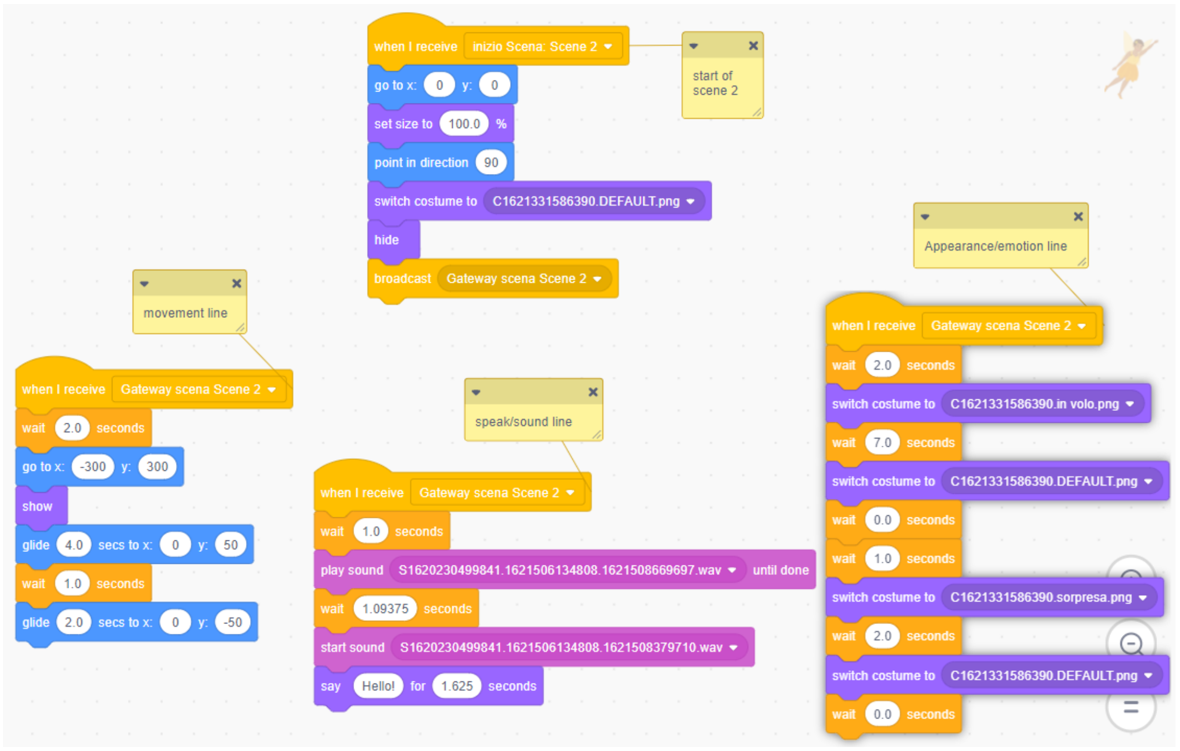


Figure 4.28 - Example: Hedgehog scene 2 IL

Scratch. In the Figure 4.29 and in the Figure 4.30, the scene 2 for "fairy" and "hedgehog" respectively are showed. For the Fairy there is anything particular to point out. For the "hedgehog" we can notice that for the movement flow empty there is only the correspondent "when I receive" Scratch block as an empty stack.



```
when I receive [inizio Scena: Scene 2 ▾]
go to x: (0) y: (0)
set size to (100.0) %
point in direction (90)
switch costume to [C1621331586390.DEFAULT.png ▾]
hide
broadcast [Gateway scena Scene 2 ▾]
```

start of scene 2

movement line

```
when I receive [Gateway scena Scene 2 ▾]
wait (2.0) seconds
go to x: (-300) y: (300)
show
glide (4.0) secs to x: (0) y: (50)
wait (1.0) seconds
glide (2.0) secs to x: (0) y: (-50)
```

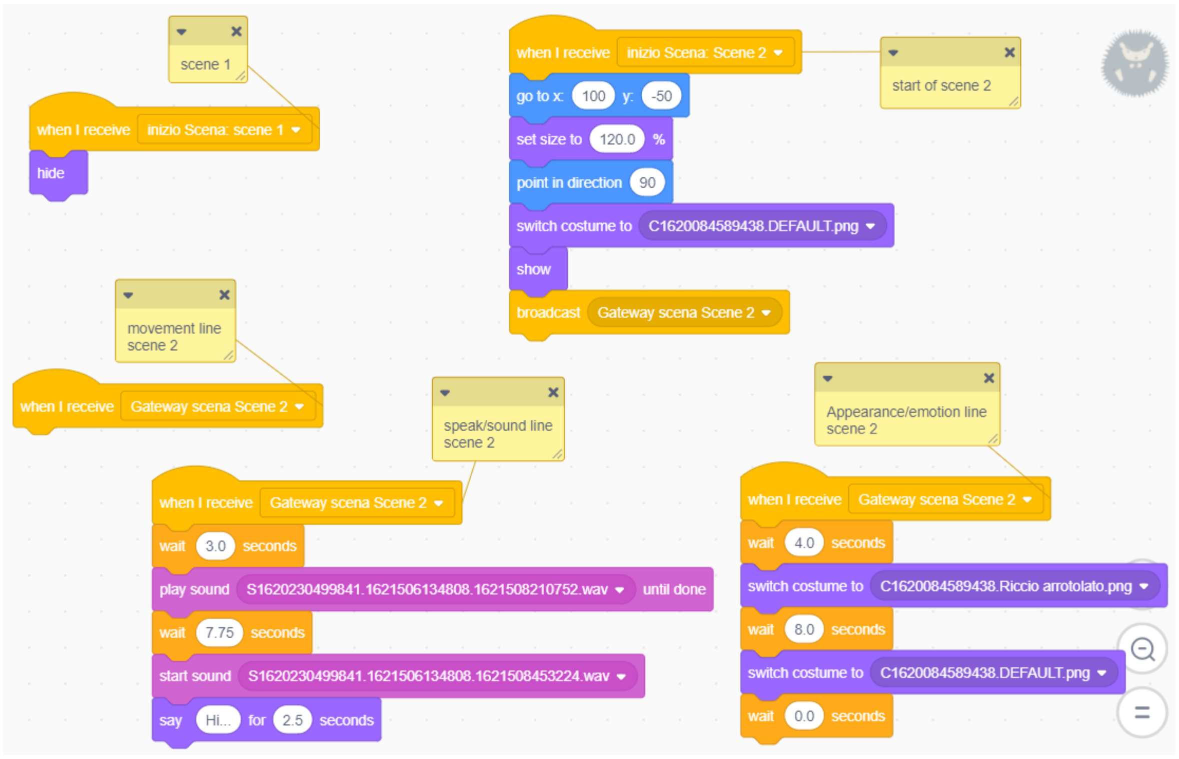
speak/sound line

```
when I receive [Gateway scena Scene 2 ▾]
wait (1.0) seconds
play sound [S1620230499841.1621506134808.1621508669697.wav ▾] until done
wait (1.09375) seconds
start sound [S1620230499841.1621506134808.1621508379710.wav ▾]
say (Hello!) for (1.625) seconds
```

Appearance/emotion line

```
when I receive [Gateway scena Scene 2 ▾]
wait (2.0) seconds
switch costume to [C1621331586390.in volo.png ▾]
wait (7.0) seconds
switch costume to [C1621331586390.DEFAULT.png ▾]
wait (0.0) seconds
wait (1.0) seconds
switch costume to [C1621331586390.sorpresa.png ▾]
wait (2.0) seconds
switch costume to [C1621331586390.DEFAULT.png ▾]
wait (0.0) seconds
```

Figure 4.29 - Example: Fairy scene 2 SCRATCH



```
scene 1

when I receive [inzio Scena: scene 1 ▾]
hide

movement line
scene 2

when I receive [Gateway scena Scene 2 ▾]

when I receive [inizio Scena: Scene 2 ▾]
go to x: (100) y: (-50)
set size to (120.0) %
point in direction (90)
switch costume to [C1620084589438.DEFAULT.png ▾]
show
broadcast [Gateway scena Scene 2 ▾]

start of scene 2

speak/sound line
scene 2

when I receive [Gateway scena Scene 2 ▾]
wait (3.0) seconds
play sound [S1620230499841.1621506134808.1621508210752.wav ▾] until done
wait (7.75) seconds
start sound [S1620230499841.1621506134808.1621508453224.wav ▾]
say (Hi...) for (2.5) seconds

Appearance/emotion line
scene 2

when I receive [Gateway scena Scene 2 ▾]
wait (4.0) seconds
switch costume to [C1620084589438.Riccio arrotolato.png ▾]
wait (8.0) seconds
switch costume to [C1620084589438.DEFAULT.png ▾]
wait (0.0) seconds
```

Figure 4.30 - Example: Hedgehog story SCRATCH

4.7 Remarks about the IL use

Using the Intermediate Language in the translation process helps to divide the translation in two separate parts. Doing this required more effort but (1) significantly reduced the complexity of the translation and (2) made it flexible and independent from both the application user interface and the execution platform.

(1) The complexity of the translation was reduced because having an intermediate step allows to reduce the “semantic distance” between the source languages and the target ones at each step. Indeed, the graphical components of the application that describe a story are more similar to the IL language than to Scratch; at the same time, the Scratch components to describe a story are more similar to the IL ones than to the application ones. Testing the code of the application during the development phase also benefited from this solution because we could test the two “parts” of the translation independently.

(2) During the development of the application, we made some modification to the user interface components and the Scratch components. Thanks to the IL, we did not have to change the entire translation process, but we modified only the affected part, that were much smaller.
At some point we needed to translate a single scene to Scratch; it was not something we originally planned to do, but it was an easy task, because we only had to create the logic for translation from the application to the IL without any need to change the logic for the translation to Scratch (because it is the same for the translation of an entire story).

5 Application User Interface

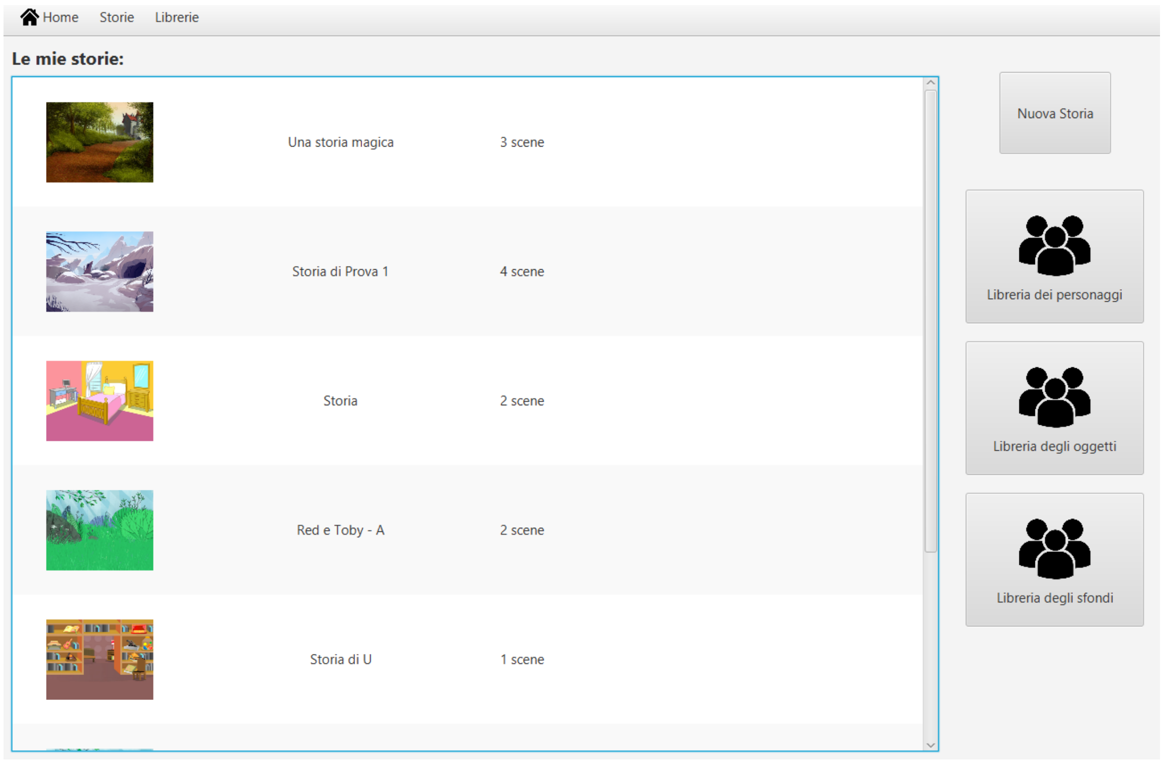
In this Section there is the description of the front-end of the application prototype, the part of the application the client interacts directly with. There are collected and described some screenshots to show the user interface of the application prototype and the available features.

5.1 Home

When the application start, the first page showed is the home page. The home page has two main panes: the list of stories and the buttons pane (see Figure 5.1)

List of stories. Each row of the list is a story, the story is represented by the background of its first scene, its title, and its number of scenes. Clicking on a row the selected story will open showing the story details (see 5.3.1 Story details).

Buttons. By clicking the first button "Nuova Storia" the user can create a new story, so the story details page will be open (see 5.3.1 Story details). The other three button are the libraries buttons (in order: characters, objects, and backgrounds), by clicking one of them the user will open the chosen library (see 5.2 Libraries)



	Una storia magica	3 scene
	Storia di Prova 1	4 scene
	Storia	2 scene
	Red e Toby - A	2 scene
	Storia di U	1 scene

Figure 5.1 - Example of Home page

5.2 Libraries

As previously described, in the application there is the possibility to add personalized characters, objects or backgrounds. Each of them has their own library, the library page has the list of the elements and a button.

List of elements. Each row of the list is the element represented by the default image and the name. Clicking on one of them the details page of the selected element will open.

Button. By clicking the button "Create a new […]" the user can create a new element, so the element details page will be open. The element details pages (one for each type of element) will be shown in the next Sections.

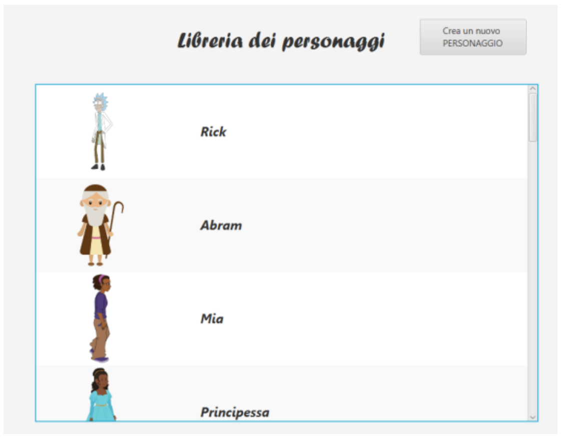


Figure 5.2 - Example of Characters Library page

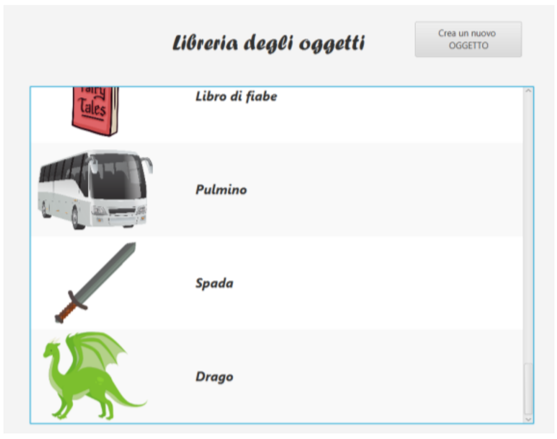


Figure 5.3 - Example of Objects Library page

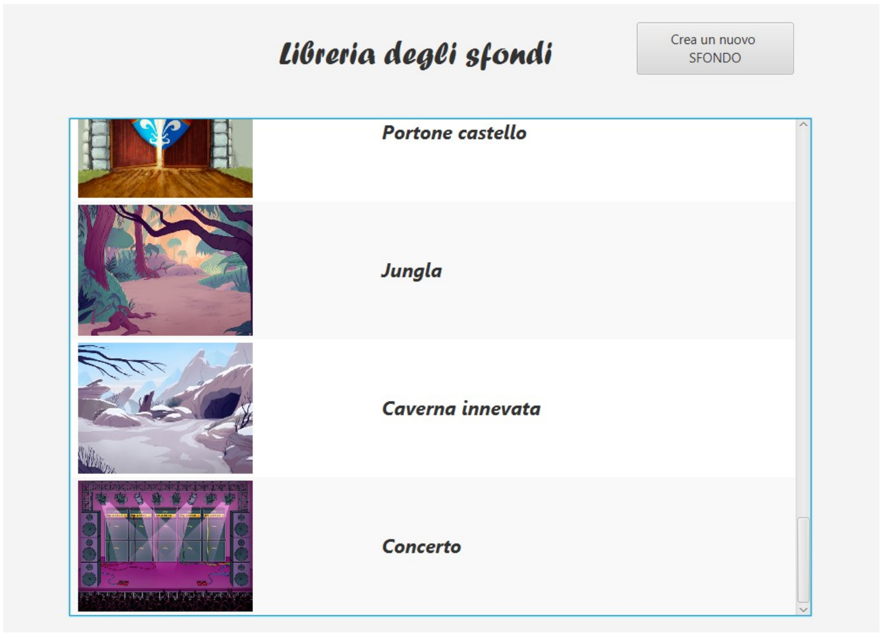


Figure 5.4 - Example of Backgrounds Library page

5.2.1 Character details

To create a new character in the character library it is mandatory to enter a name and a default image choosing a file from the pc, the other fields are optional. The details fields is where the user can add some information about the character. If the user does not enter an image for the emotion fields, the application will use the default one. The personalized images list contains, for each row, the image, the name of the image and a button to delete the image from the character fields. Below the personalized images list there is a button to add a new personalized image, clicking on the button opens a pop-up where the user can enter the image name and the image file choosing a file from the pc.

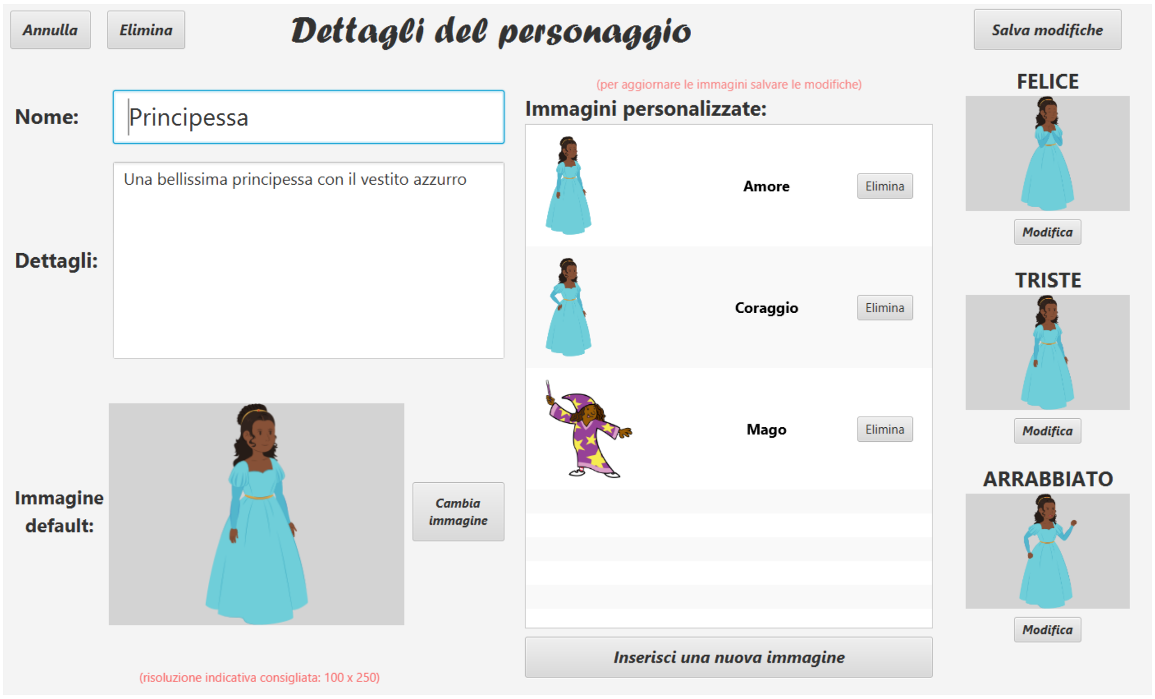
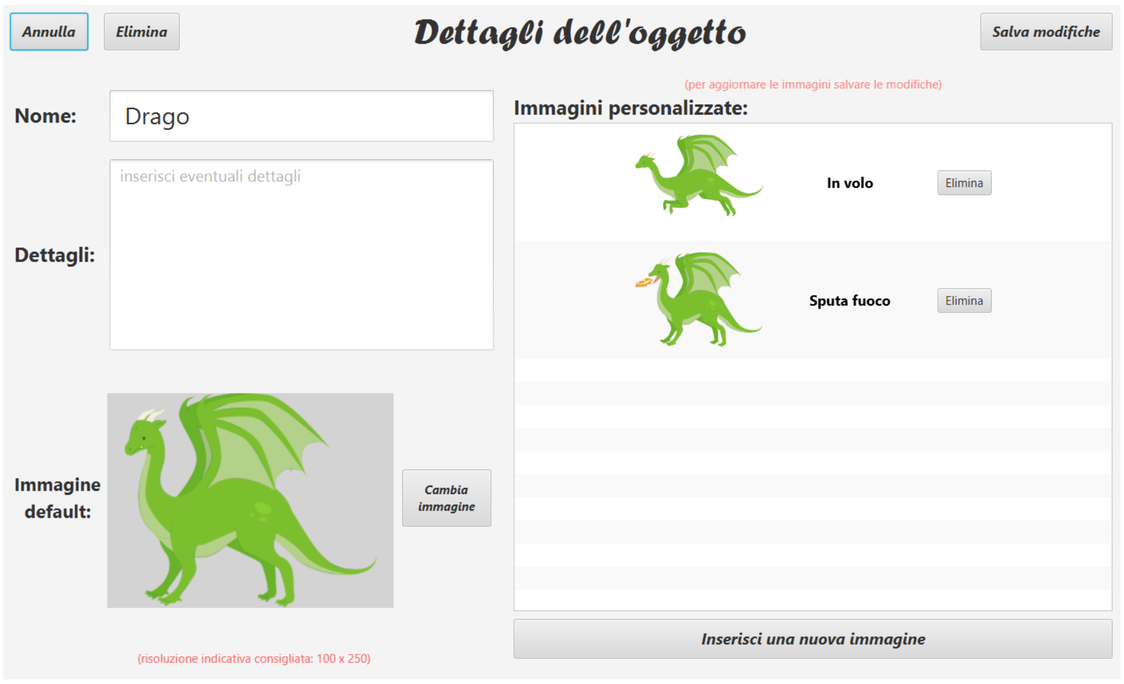


Figure 5.5 - Example of Character Details page

5.2.2 Object details

The object details page is the same of the character ones described in the previous Section, only without the emotions fields.



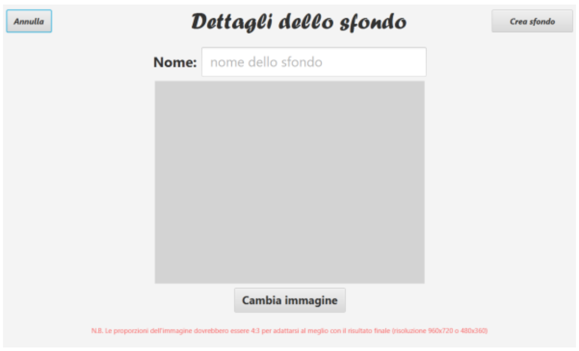
Nome: Drago

Dettagli: inserisci eventuali dettagli

Figure 5.6 - Example of Object Details page

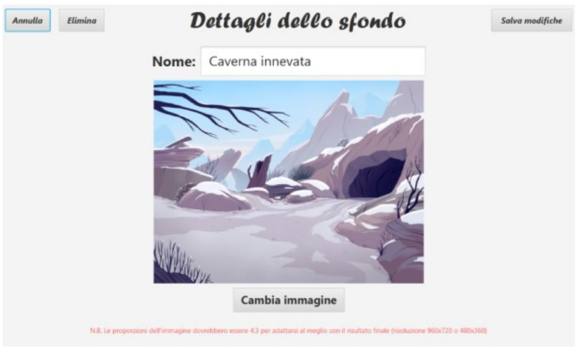
5.2.3 Background details

The backgrounds details page has only the name field and the image of the background. To create the new image, the user needs to enter a name and an image file from the pc. User can create a new Background or modify an existing one.


Annulla
Dettagli dello sfondo
Crea sfondo
Nome:
Cambia immagine
N.B. Le proporzioni dell'immagine dovrebbero essere 4:3 per adattarsi al meglio con il risultato finale (risoluzione 960x720 o 480x360)

Nome: nome dello sfondo

Figure 5.7 - Create a new Background page


Annulla
Elimina
Dettagli dello sfondo
Salva modifiche
Nome:
Cambia immagine
N.B. Le proporzioni dell'immagine dovrebbero essere 4:3 per adattarsi al meglio con il risultato finale (risoluzione 960x720 o 480x360)

Nome: Caverna innevata

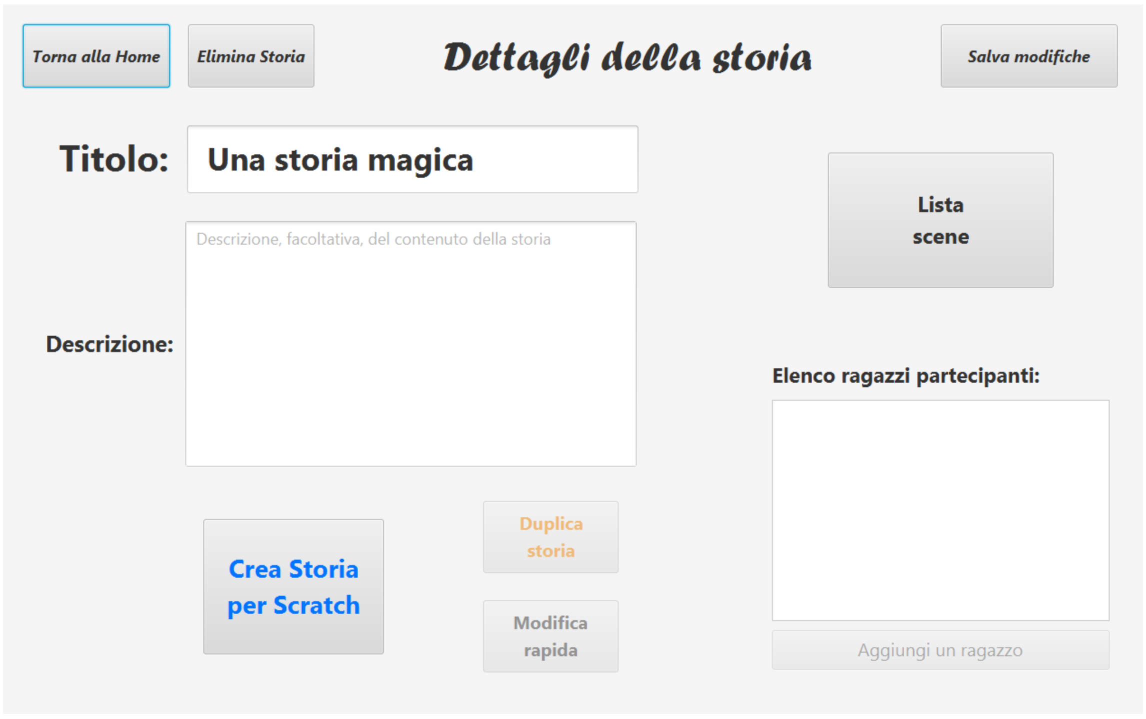
Figure 5.8 - Example of Background details page

5.3 Story

While the libraries are the Section of the application where the user can create and modify Characters, Objects and Backgrounds, the Story pages are where the user can import them from libraries and create a personalized story. In the next Sections we will show all the pages useful to create or modify a story.

5.3.1 Story details

The story details page has the Title field that is mandatory for the creation of a new story, and the description field that is optional. With the “Lista scene” button the user can open the List of scenes of the story. The “Create Story for Scratch” button create a new .sb3 file of the story in the directory selected from the user. The three buttons in the top Section of the page are: come back to home, delete story, and save changes. There are other three disabled buttons that represent additional features that we would like to add that are not yet been fully implemented, they will be described in the Section 8.2.



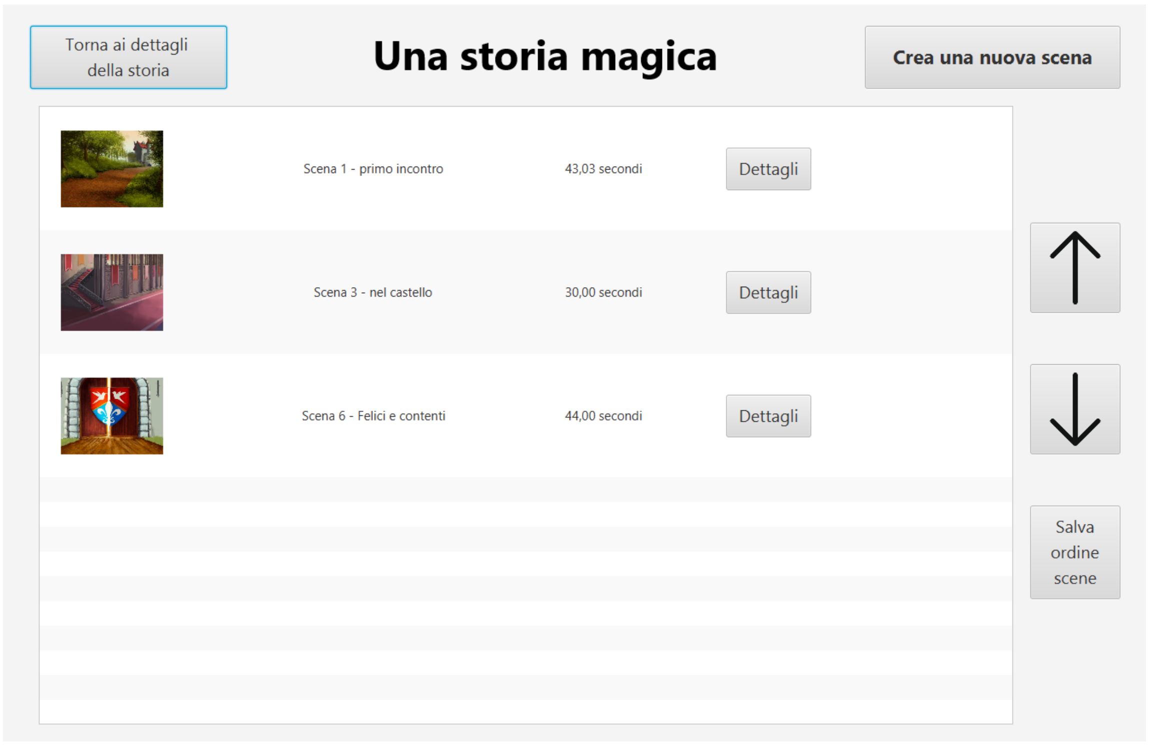
Titolo: Una storia magica

Descrizione: Descrizione, facoltativa, del contenuto della storia

Figure 5.9 – Example of Story Details page

5.3.2 List of scenes of the story

On the top of the page there is the "come back to story details" button that open the *Story details* page, the title of the selected story and the "create new scene" button that open the Scene details page for the creation of a new scene. In the center of the page there is the list of the scenes, where, by clicking on a single scene, the user can change its position in the list with the up and down arrow in the right side of the page. For every row of the list there is a button to open the Scene details and some information of the scene: the background, the title of the scene and the duration in seconds.



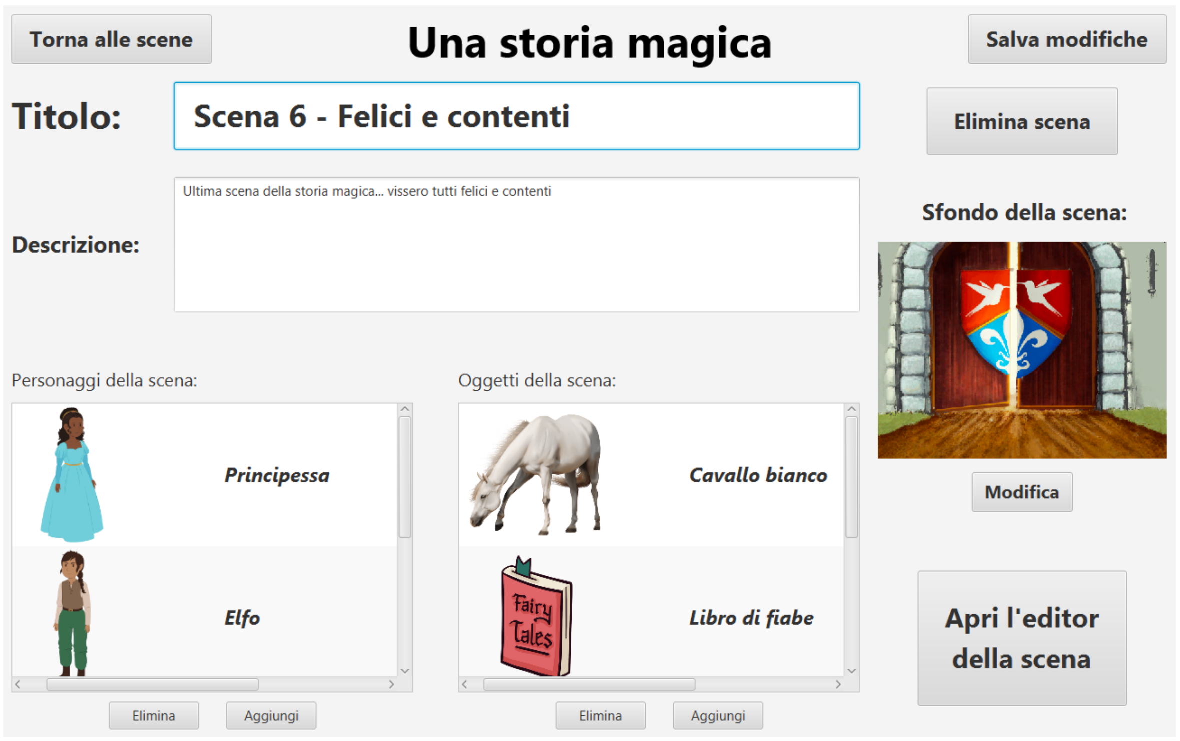
Una storia magica

	Scena 1 - primo incontro	43,03 secondi	Dettagli
	Scena 3 - nel castello	30,00 secondi	Dettagli
	Scena 6 - Felici e contenti	44,00 secondi	Dettagli

Figure 5.10 - Example of Scenes List page

5.3.3 Scene details

On the top of the page there is the "Come back to scenes" button that open the *List of scenes of the story* page, the title of the selected story, the "Save changes" button that save the changes in the scene fields, and, below, the "Delete scene" that delete the scene and open the *List of scenes of the story* page. On the bottom right corner there is the "Open the scene editor" button that open the Scene Editor page. To create a new scene of the story it is mandatory to enter a title and a background image choosing a file from the pc, the other fields are optional. The optional fields are: a description of the content of the scene, the list of characters and the list of objects that are present in the scene. Background, Characters and Objects can be chosen and imported from the libraries of the application clicking on the "edit" button for the first and on the "add" button for the others two.


Una storia magica

Titolo: Scena 6 - Felici e contenti

Descrizione: Ultima scena della storia magica... vissero tutti felici e contenti

Personaggi della scena: Principessa, Elfo

Oggetti della scena: Cavallo bianco, Libro di fiabe

Sfondo della scena:

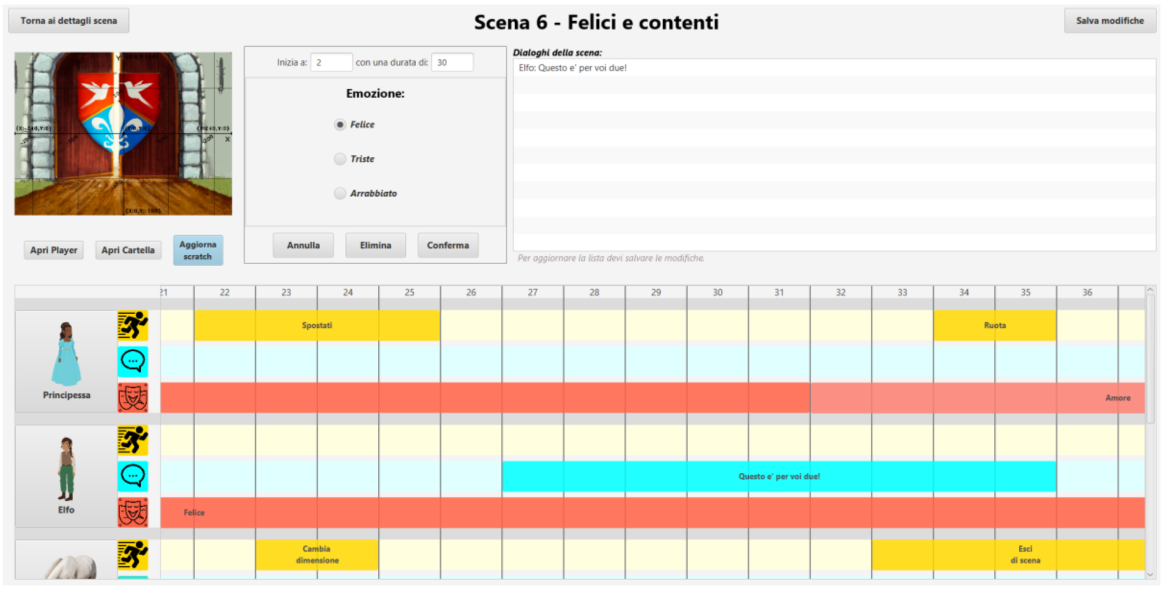
Figure 5.11 - Example of Scene Details page

5.3.4 Scene Editor

The more complex Section of the application is the scene editor. Here the users of the application can interact with a timeline and create the content of a scene, which will be represented by some initial settings and a sequence of blocks (described in the Section 4.3.2). There are two examples at the end of this Section: Figure 5.12 and Figure 5.13.

On the top of the page there is the “Come back to scene details” button that open the *Scene details* page, the title of the selected scene, the “Save changes” button that save the changes.

Below there is a horizontal pane divided in three part: (1) The background image with a coordinate plane on it (that help to find a specific point x, y in the image). (2) The selected block details: where all the details of the block are shown and where the user can enter or change the fields which vary according to the selected block. (3) The list of the lines of the scene with the name of the character followed by the text of the line.



Scena 6 - Felici e contenti

Inizia a: 2 con una durata di: 30

Emozione:

[✓ Felice]

Triste

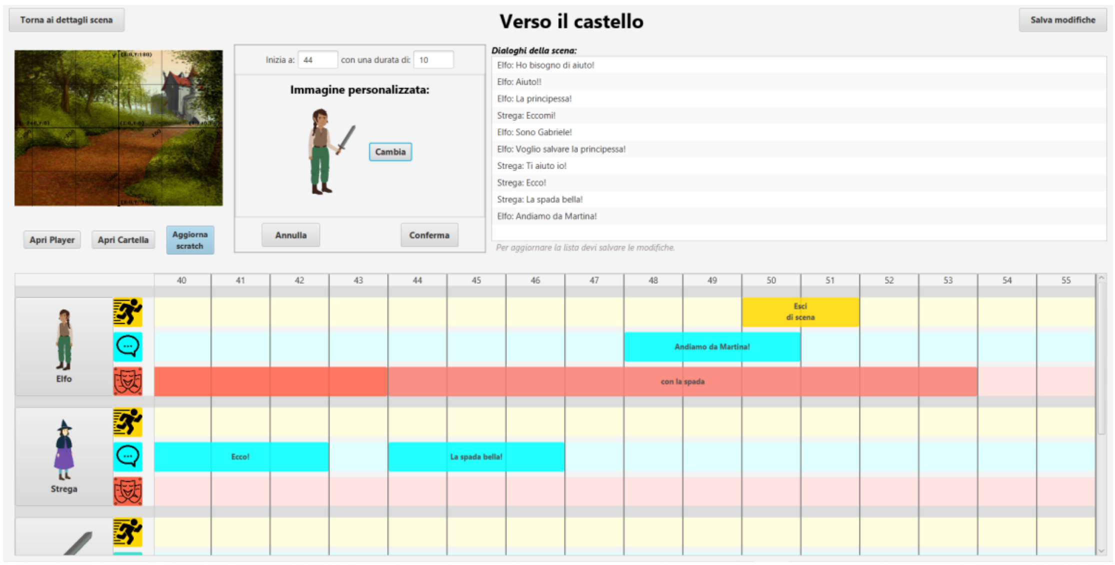
Arrabbiato

Dialoghi della scena:

Elfo: Questo e' per voi due!

Per aggiornare la lista devi salvare le modifiche.

Figure 5.12 – Example 1 of Scene Editor page



Verso il castello

Inizia a: 44 con una durata di: 10

Immagine personalizzata:

Dialoghi della scena:

Elfo: Ho bisogno di aiuto!

Elfo: Aiuto!!

Elfo: La principessa!

Strega: Eccomi!

Elfo: Sono Gabriele!

Elfo: Voglio salvare la principessa!

Strega: Ti aiuto io!

Strega: Ecco!

Strega: La spada bella!

Elfo: Andiamo da Martina!

Per aggiornare la lista devi salvare le modifiche.

Figure 5.13 – Example 2 of Scene Editor page

The main section of the editor page is the Timeline divided vertically into seconds which is composed as follows:

A section for each **character** consisting of:

- Actions (yellow): within which it is possible to manage the movement, rotation, zoom, scene entry/exit of the character.
- Dialogs (cyan) and sounds (turquoise): to allow the character to speak or to make sounds.
- Emotions (light red) and appearance (soft red): where it is possible to manage the emotion or the look of the character.

A section for each **object** of the scene divided into:

- Actions (yellow): same as that of the character.
- Sounds (turquoise): to insert sounds associated with the object.
- Appearance(soft red): where it is possible to manage the look of the object.

A section for **narration** (turquoise).

A section for **background music** (turquoise).

A section for **sound effects** (turquoise).

6 Implementation

For the implementation of the entire thesis, we used NetBeans IDE 8.2, with Java 1.8, and, for the front-end, we used JavaFx, with FXML. In the next chapter we will see some details about the implementation.

6.1 Data Model

We want to present the data model of the Application, and the data model based on Scratch file format. These two parts are the source model and the target model of the translation, in the middle there is the Intermediate Language, the data model of this one is already showed in the Section 3.5.

6.1.1 App data model

In Figure 6.1 and Figure 6.2 the app data model is showed. We omitted the class "Person" because we are not using it (more details in Section 8.2) and the getter and setter of all the attributes. The coloured classes are the most important to give meaning to the data model.

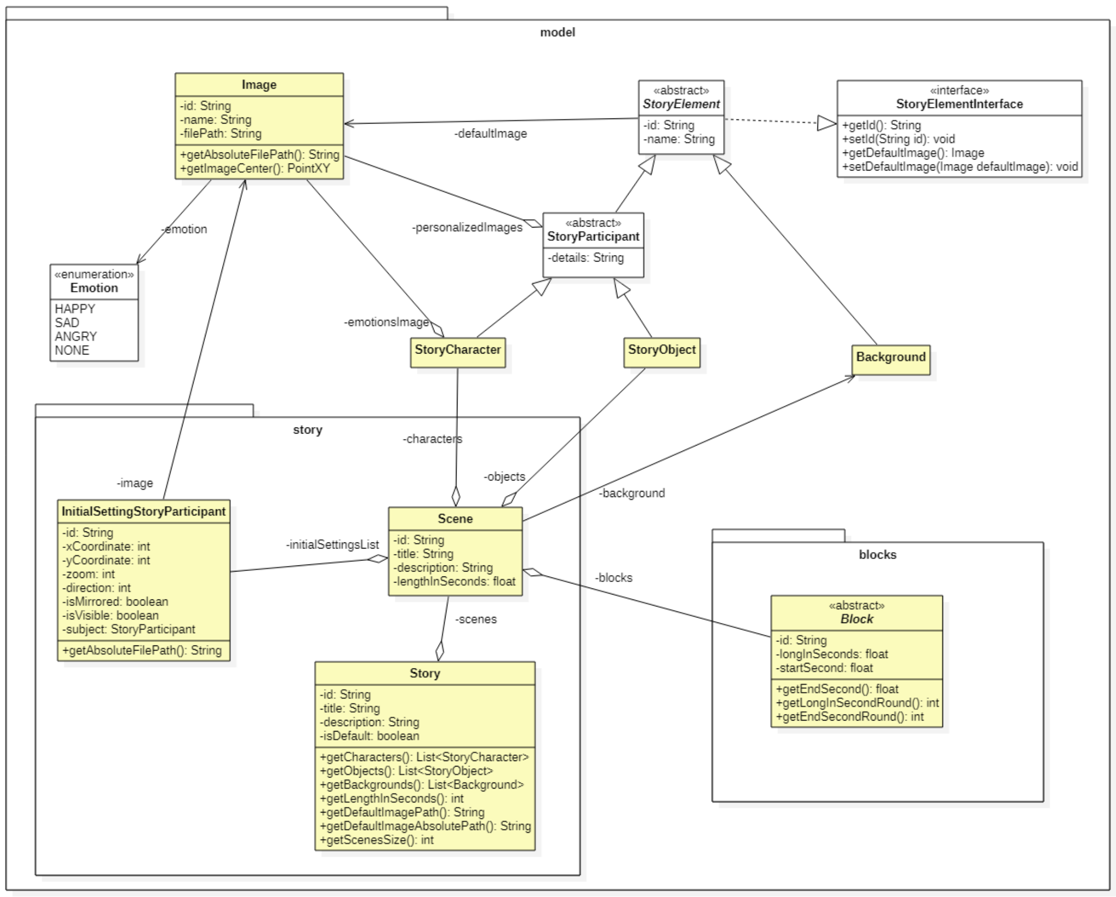


Figure 6.1 – UML class diagram: App data model

In this paragraph we described some interesting information that are not showed in the UML. The user has to choose characters, objects, and background of the scene from the ones in the libraries (as previously said the application has the libraries of the elements and the list of all the story as two independent part), and the chosen element is imported into the scene, so from then on, they will "live" as two separate objects in memory. The Emotion enumeration has the override of the *toString* method that return the Italian version of the enumeration values.

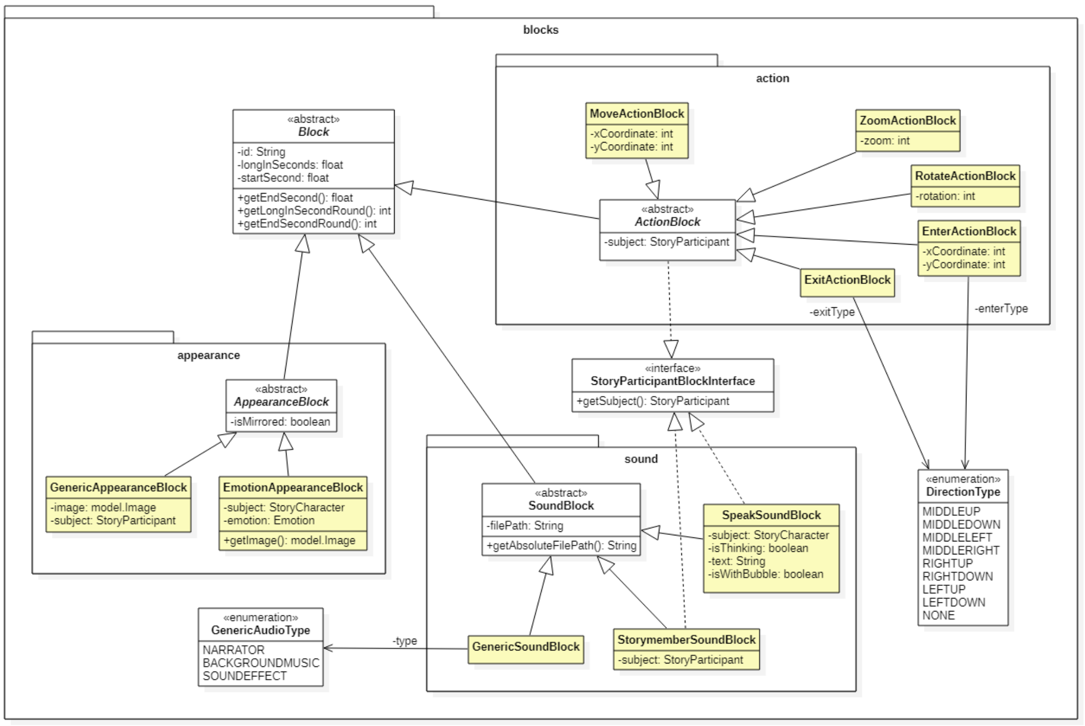


Figure 6.2 - UML class diagram: Blocks App data model

The method *getImage* of the EmotionAppearanceBlock return the default image of the "subject" if the user does not add an image for the current emotion. In the "helper" package, there is a class that check if the times data for one block are valid with respect to another, in particular given two blocks, or given one block + the start and finish times of another block, or given the start and finish times of the blocks, it will return *true* if the two blocks "overlap" and *false* otherwise (the validation of the block times are based on the idea of timeline).

Every time one story is edited, the json file of that story is updated.

In the helper package there is also the "StoryInfo" class that is useful for the home page. In fact, when we open the app on the homepage, none of the stories are in memory, but only the list of StoryInfo, where there are only the useful information of the stories for the stories list creation.

6.1.2 Scratch data model

In our case the target language of the translation is Scratch 3.0 file format, that is the format used to store exported Scratch 3.0 projects. These are ZIP archives, with the extension .sb3, which contain information encoded in a text-based format called JSON and project media in separate files.[22]

An .sb3 file is a ZIP archive containing one JSON file, *project.json*, representing the project.

The following UML data model is useful to describe the *project.json* file format[23].

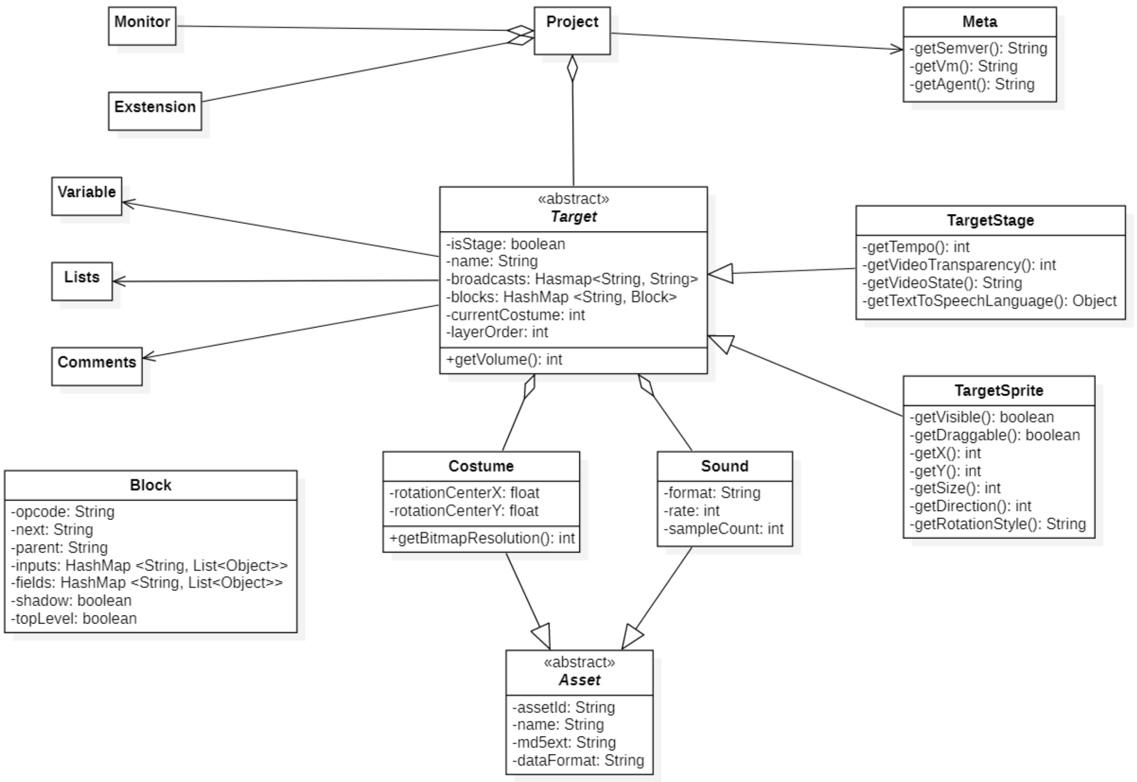


Figure 6.3 - UML class diagram: Scratch data model

The Jackson `ObjectMapper.writeValueAsString(Project project)` automatically create the String for the *project.json* file.

There are some empty classes, these do not have any attribute nor methods because are some Scratch features not useful for our case study and in the Scratch file format can be an empty value.

[22] https://en.scratch-wiki.info/wiki/Scratch_File_Format
[23] https://en.scratch-wiki.info/wiki/Scratch_File_Format#Format

All the private attributes have their getter and setter that we omitted for simplicity.

There are some classes with only getter methods (also the Costume class has one): these return some fixed values that we want that the Jackson ObjectMapper fills automatically. The fixed values are some Scratch default values or some values depending on our use case.

6.2 App resources

The stories and the libraries are saved on files with the helper of Jackson. In the Figure 6.4 there is an example of the application data on files organization: folders, json files and images files (represented with a circle on the top of the file name). The files name and the story names are the ID of the corresponding object in memory. The ID is created with the *System.currentTimeMillis()* preceded by a letter (B for backgrounds, C for characters, O for objects, S for stories), in the case of Image objects, the ID is the ID of the element (background, object or character) followed by: "DEFAULT" (if it is a default image), the EMOTION (if it is an emotion image, only characters can have one), or the name of the personalized image (only characters and objects can have one).

The json files in "storymemberlibrary" contain the list of the corresponding elements of the app. The *storiesList.json* contain the list of "StoryInfo" for each story (as previously said, "storyInfo" contains some main information of the story).

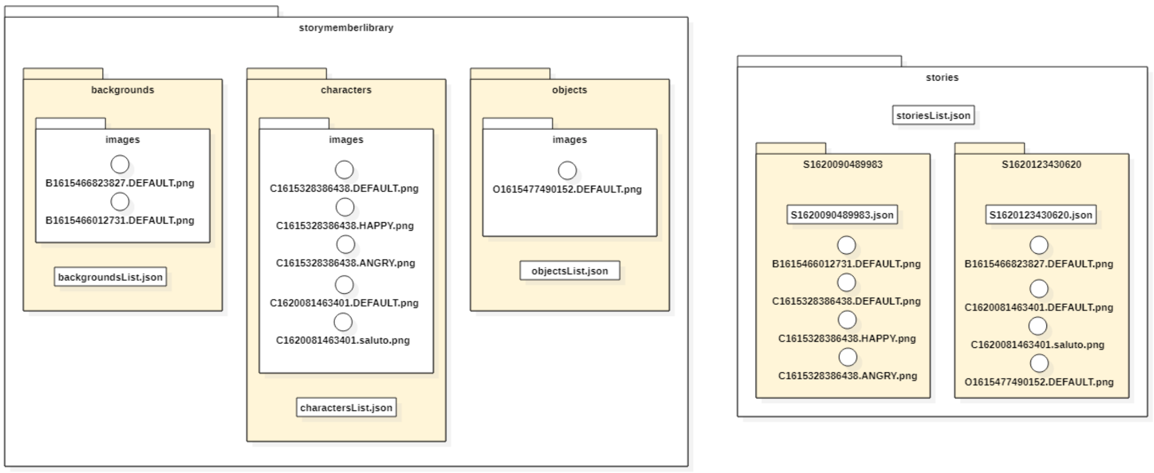

storymemberlibrary
backgrounds
images
B1615466823827.DEFAULT.png
B1615466012731.DEFAULT.png
backgroundsList.json
characters
images
C1615328386438.DEFAULT.png
C1615328386438.HAPPY.png
C1615328386438.ANGRY.png
C1620081463401.DEFAULT.png
C1620081463401.saluto.png
charactersList.json
objects
images
O1615477490152.DEFAULT.png
objectsList.json
stories
storiesList.json
S1620090489983
S1620090489983.json
B1615466012731.DEFAULT.png
C1615328386438.DEFAULT.png
C1615328386438.HAPPY.png
C1615328386438.ANGRY.png
S1620123430620
S1620123430620.json
B1615466823827.DEFAULT.png
C1620081463401.DEFAULT.png
C1620081463401.saluto.png
O1615477490152.DEFAULT.png

Figure 6.4 - App data files example

7 Results

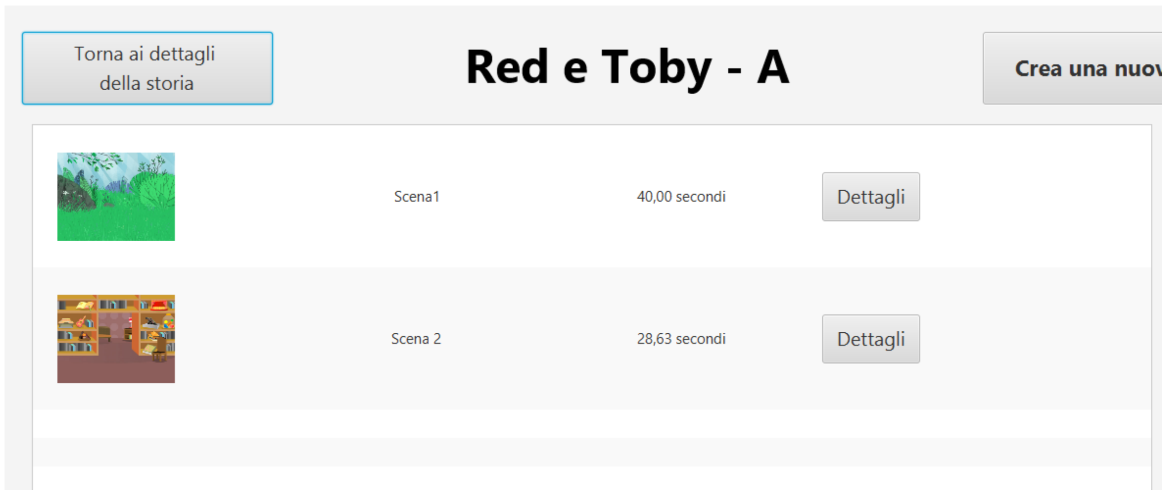
A The application's beneficial results will be proved from a practical point of view through the testing of the prototype with Esagramma[1]. First, we will underline the results obtained regarding the translation of the story from the application block language into the Scratch block graphical representation. Then we will analyze the results of the interaction of the participants from the associations with the application.

7.1 Results of the translation

We present the result of the creation of the story during the experiments focusing on only one of the main characters. In the next Sections we show the scene 1 and 2 from the point of view of one of the main characters ("Red") in the customized language of the app and in the Scratch block language. Here we will show only the results, there is an example of the translation process in details in the Section 4.6.

7.1.1 Example A: story

In the Figure 7.1 we can see the App view of the list of scenes of the story. The story has two scenes: "Scena 1" of 40 seconds and "Scena 2" of 28,63 seconds each. The background of the first scene is the "forest", the background of the second scene is the "room".



	Scena1	40,00 secondi	Dettagli
	Scena 2	28,63 secondi	Dettagli

Figure 7.1 - App scenes list: Example A

The Figure 7.2 displays the result of the story evolution after the translation into the Scratch language, from the Stage element of Scratch point of view.

The "when flag clicked" block shows the "start" of the story. Then the second block set the scene background image as the "forest" one. It is followed by the "broadcast" block that indicates that the first scene is “started”. Once we arrived to the “wait” block with the duration of 40 seconds, which represents the whole duration of scene 1, the procedure will be repeated for the second scene.

The blocks on the right are all empty stack (with only one or two “received” or “broadcast” blocks) because there are not any narration/background music/sound effects in the story.

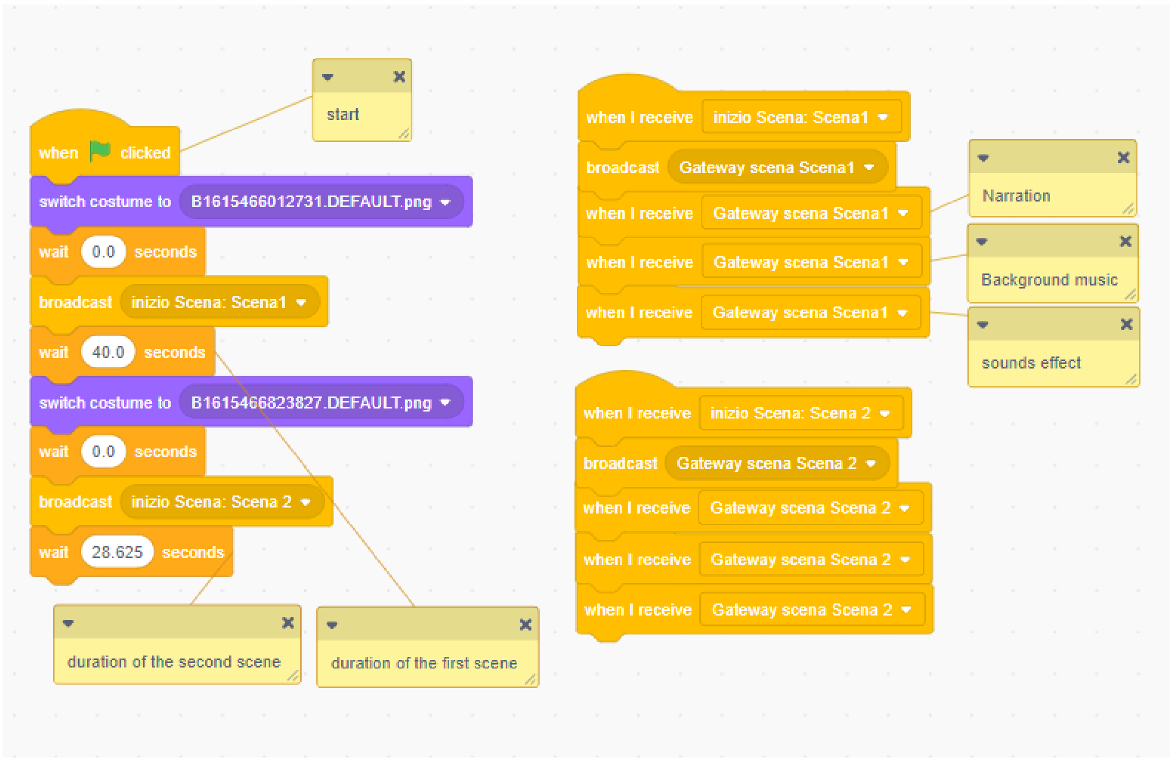

```
when 🏴 clicked
switch costume to B1615466012731.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: Scena1
wait 40.0 seconds
switch costume to B1615466823827.DEFAULT.png
wait 0.0 seconds
broadcast inizio Scena: Scena 2
wait 28.625 seconds

when I receive inizio Scena: Scena1
broadcast Gateway scena Scena1
when I receive Gateway scena Scena1
when I receive Gateway scena Scena1
when I receive Gateway scena Scena1

when I receive inizio Scena: Scena 2
broadcast Gateway scena Scena 2
when I receive Gateway scena Scena 2
when I receive Gateway scena Scena 2
when I receive Gateway scena Scena 2
```

Figure 7.2 - Scratch blocks: Story Example A

7.1.2 Example A: scene 1

In the Figure 7.3 there is the timeline for the character “Red” in the scene 1.

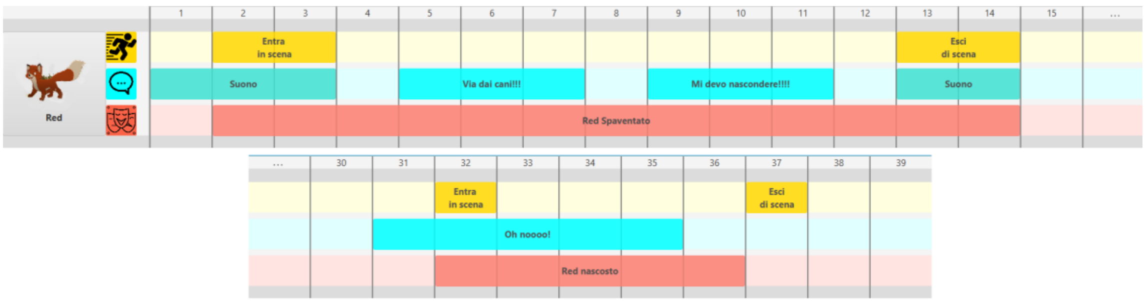
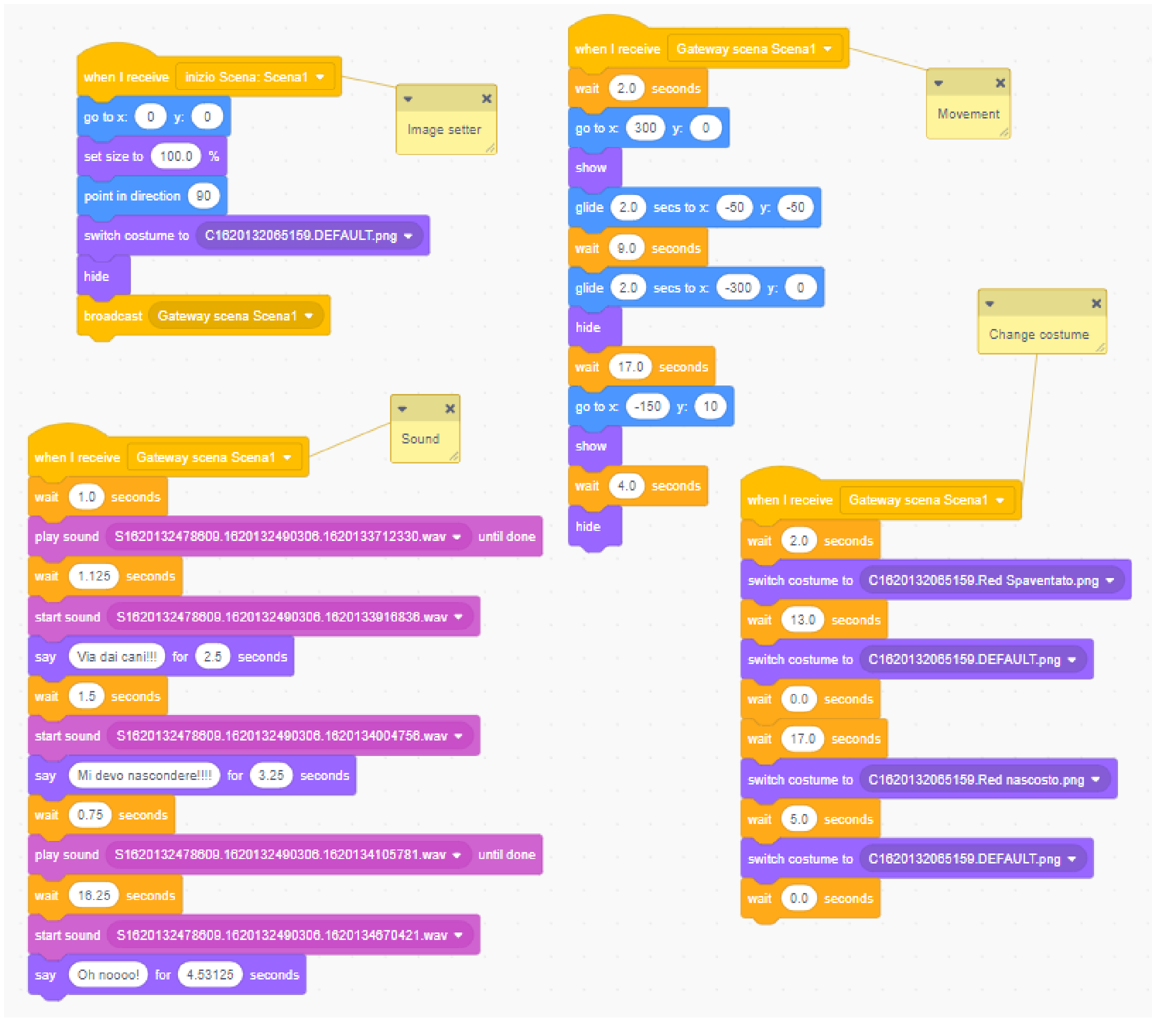


Figure 7.3 - App blocks: scene 1 Example A

The Figure 7.4 shows the result of the translation for the “Red” character in Scene 1.



```
when I receive [inizio Scena: Scena1 ▾]
go to x: (0) y: (0)
set size to (100.0) %
point in direction (90)
switch costume to [C1620132065159.DEFAULT.png ▾]
hide
broadcast [Gateway scena Scena1 ▾]

when I receive [Gateway scena Scena1 ▾]
wait (1.0) seconds
play sound [S1620132478609.1620132490306.1620133712330.wav ▾] until done
wait (1.125) seconds
start sound [S1620132478609.1620132490306.1620133916836.wav ▾]
say (Via dai cani!!!) for (2.5) seconds
wait (1.5) seconds
start sound [S1620132478609.1620132490306.1620134004756.wav ▾]
say (Mi devo nascondere!!!!) for (3.25) seconds
wait (0.75) seconds
play sound [S1620132478609.1620132490306.1620134105781.wav ▾] until done
wait (16.25) seconds
start sound [S1620132478609.1620132490306.1620134670421.wav ▾]
say (Oh noooo!) for (4.53125) seconds

when I receive [Gateway scena Scena1 ▾]
wait (2.0) seconds
go to x: (300) y: (0)
show
glide (2.0) secs to x: (-60) y: (-60)
wait (9.0) seconds
glide (2.0) secs to x: (-300) y: (0)
hide
wait (17.0) seconds
go to x: (-150) y: (10)
show
wait (4.0) seconds
hide

when I receive [Gateway scena Scena1 ▾]
wait (2.0) seconds
switch costume to [C1620132065159.Red Spaventato.png ▾]
wait (13.0) seconds
switch costume to [C1620132065159.DEFAULT.png ▾]
wait (0.0) seconds
wait (17.0) seconds
switch costume to [C1620132065159.Red nascosto.png ▾]
wait (5.0) seconds
switch costume to [C1620132065159.DEFAULT.png ▾]
wait (0.0) seconds
```

Figure 7.4 –Scratch blocks: scene 1 Example A

7.1.3 Example A: scene 2

In the Figure 7.5 there is the timeline for the character “Red” in the scene 2.

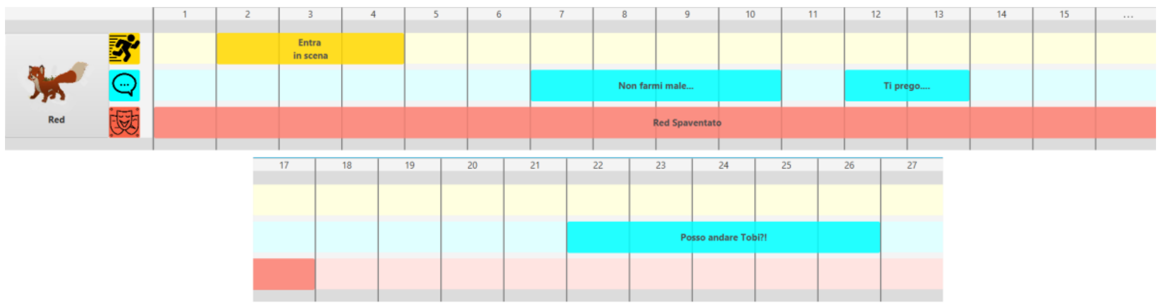
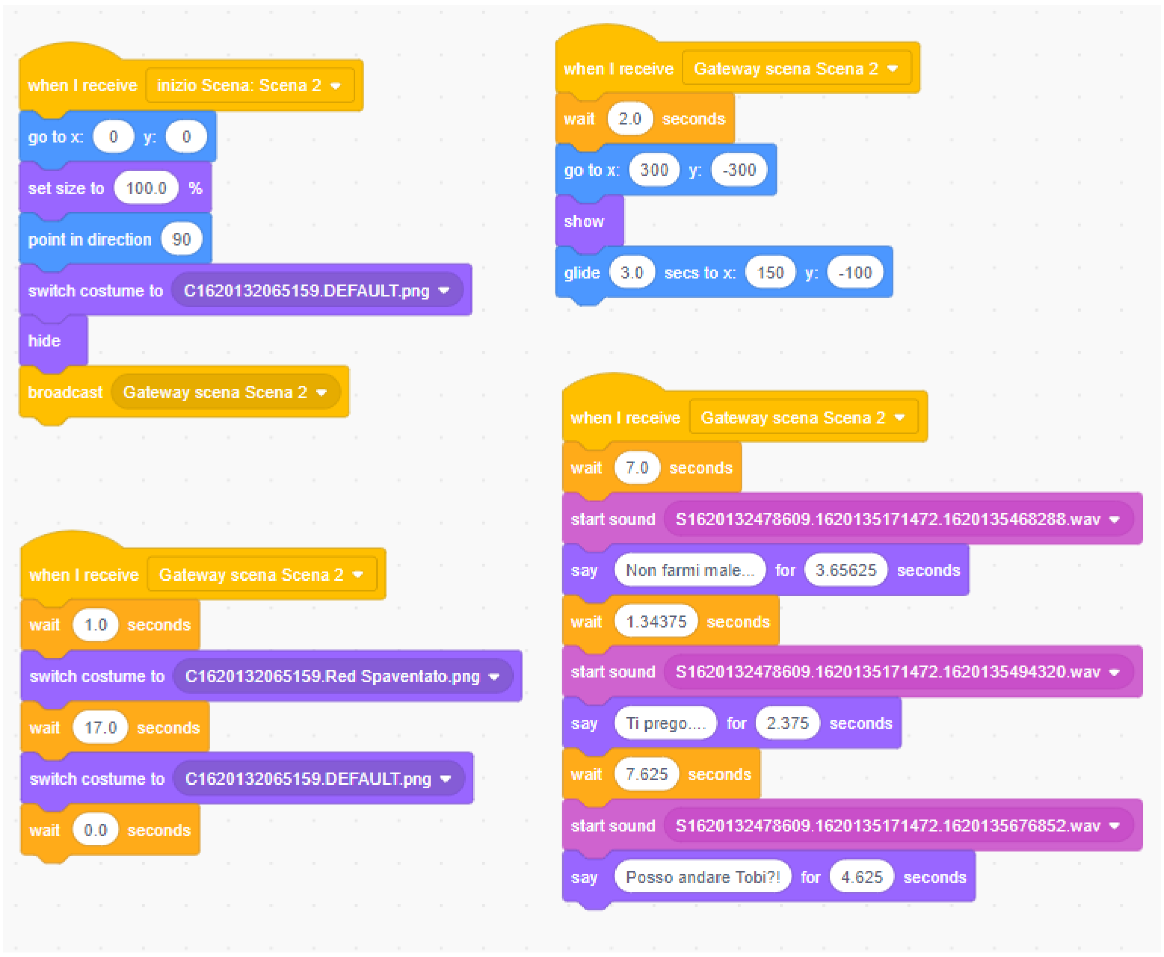


Figure 7.5 - App blocks: scene 2 Example A

The Figure 7.6 shows the result of the translation for the “Red” character in Scene 2.



```
when I receive [inizio Scena: Scena 2 ▾]
go to x: (0) y: (0)
set size to (100.0) %
point in direction (90)
switch costume to [C1620132065159.DEFAULT.png ▾]
hide
broadcast [Gateway scena Scena 2 ▾]
```

```
when I receive [Gateway scena Scena 2 ▾]
wait (1.0) seconds
switch costume to [C1620132065159.Red Spaventato.png ▾]
wait (17.0) seconds
switch costume to [C1620132065159.DEFAULT.png ▾]
wait (0.0) seconds
```

```
when I receive [Gateway scena Scena 2 ▾]
wait (2.0) seconds
go to x: (300) y: (-300)
show
glide (3.0) secs to x: (150) y: (-100)
```

```
when I receive [Gateway scena Scena 2 ▾]
wait (7.0) seconds
start sound [S1620132478609.1620135171472.1620135468288.wav ▾]
say (Non farmi male...) for (3.65625) seconds
wait (1.34375) seconds
start sound [S1620132478609.1620135171472.1620135494320.wav ▾]
say (Ti prego....) for (2.375) seconds
wait (7.625) seconds
start sound [S1620132478609.1620135171472.1620135676852.wav ▾]
say (Posso andare Tobi?!) for (4.625) seconds
```

Figure 7.6 –Scratch blocks: scene 2 Example A

7.2 Testing of application prototype: Esagramma

The prototype testing was conducted with the Fondazione Sequeri Esagramma Onlus. It is an organization in Milan that has been operating for over 30 years. Its services range from offering rehabilitation to therapeutic and artistic paths aimed at inclusion, personal growth, and the rediscovery of beauty in all of its forms especially in music, for people with disabilities.

In the first Section, the test settings are described, in the second Section the description of the different participants and the results of the experiments will be provided. Then we will present the feedback of two educators who assisted during the session, Cecilia Pagliardini[24] and Cristina Bulgheroni[25].

7.2.1 Test settings

To evaluate the effectiveness of the application, an experiment of few weeks was planned, involving a group of young people from the association. Unfortunately, for various reasons, it was only possible to conduct only one session in which one hour was dedicated for each of the four participants of moderate/severe intellectual disability.

The test was made with the support of a computer and an interactive whiteboard (IWB). The best setting in terms of user experience turned out to be the following: showing the editor on the computer and the player of the scene/story on the IWB (see Figure 7.8 for an example of the best setting).

The participants were:

- A: male, 14 years old, Down syndrome,
- M: female, 63 years old, Down syndrome,
- U: male, 30 years old, severe General Learning Disability,
- G: male, 20 years old, mental retardation and autistic traits.

The purpose of the testing phase was to evaluate the application from three aspects: (1) the satisfaction of the user with the application, (2) the effectiveness it has in

[24] Clinical psychologist, cognitive-constructivist psychotherapist. Master on Esagramma methodologies. She has been collaborating since 2013 as a psychologist and music therapist with Esagramma, where she leads groups of MusicoTerapiaOrchestrale® and individual paths of MultiMedial Interaction.

[25] She obtained the Master as an expert in musical and multimedia paths for frailty at Esagramma. She has been working in Esagramma since 2014, where she is part of the in MusicoTerapiaOrchestrale® groups, she is educator in the affective vocal education project, and she leads individual paths of MultiMedial Interaction.

helping the user with intellectual disability improve his/her communication skills and (3) the application's performance levels and error occurrence.

7.2.2 Description of results

We followed different approaches with the four participants, thus obtaining different results that will be highlighted in this Section. The results are in accordance with the point of view and feedback of the educators (see the tables in the appendix 2-5, the table used is a modified version of the one from the LYV project).

Subject A, assisted by Cecilia Pagliardini. The subject of the first experiment is a 14-years-old boy with Down syndrome. The story of this test is predefined, but it is A who chooses, guided by educators, which lines to have the characters say.

The problems of concentration and communication, anticipated by the educator, are evident. In the last sessions of the multimedia laboratory, he is working on "The Fox and the Hound" tale ("Red e Toby"). Thus, the educator decided to set up the experiment based on this tale. Together with the educators, 30 minutes before the start of the session, we have decided which scenes of the tale we want to complete with A, and we created the two main characters. The experiment was based on the ability of the user to represent the main characters in the scene that he previously studied, to come up with the lines of the dialogue in the scene, record his voice trying to express the emotions of the characters, and to make sounds representing the actions in the scene (for example loud and fast knock on the table for running noise or slow and soft tap on bongo for soft footsteps noise).

The results obtained were satisfying, the use of the application did not bore him as he felt very happy to see the characters move and speak with his voice. Moreover, despite his concentration problems, he was always very focused and involved, even if this required a short break in the middle of the experiment as he was not used to concentrate for so long during his interaction with strangers.

Narration. N.A.

Lines. There was a constant evolution of the vocal expression and the emotion expression during registration of the lines, but at the end of the last scene there was a decline in attention and vocal expression because the subject was really tired due to the high value of concentration required during the exercise.

Subject M, assisted by Cecilia Pagliardini. The protagonist of the second experiment is a 63-years-old woman with Down syndrome. The story made during this test is invented by the participant. She has chosen the main characters, but the settings were already prepared.

She finds hard difficulties in speaking and communicating. In the test phase, the user appears active and engaged with a good ability to create and follow the storyline. However, she struggles to express herself as she needs assistance for speaking and requires a lot of time to tell the story she has decided to perform during the test, but, at the end, she was really satisfied with the story she had created.

It is important to underline that the subject showed significant progress during the experiment as her vocal participation considerably improved.

Narration. At the last scene of the story, she tried to narrate the events of the scene together with the educator, while previously she would take a long time to explain to us what she wanted to happen in the story and would let the educator narrate the scenes without even trying herself.

Lines. At the last scene of the story, the line she registered was really good in comparison with those of the first scene. In fact, she was able to record the line entirely, without having to take a break in between words and without being helped by the educator.

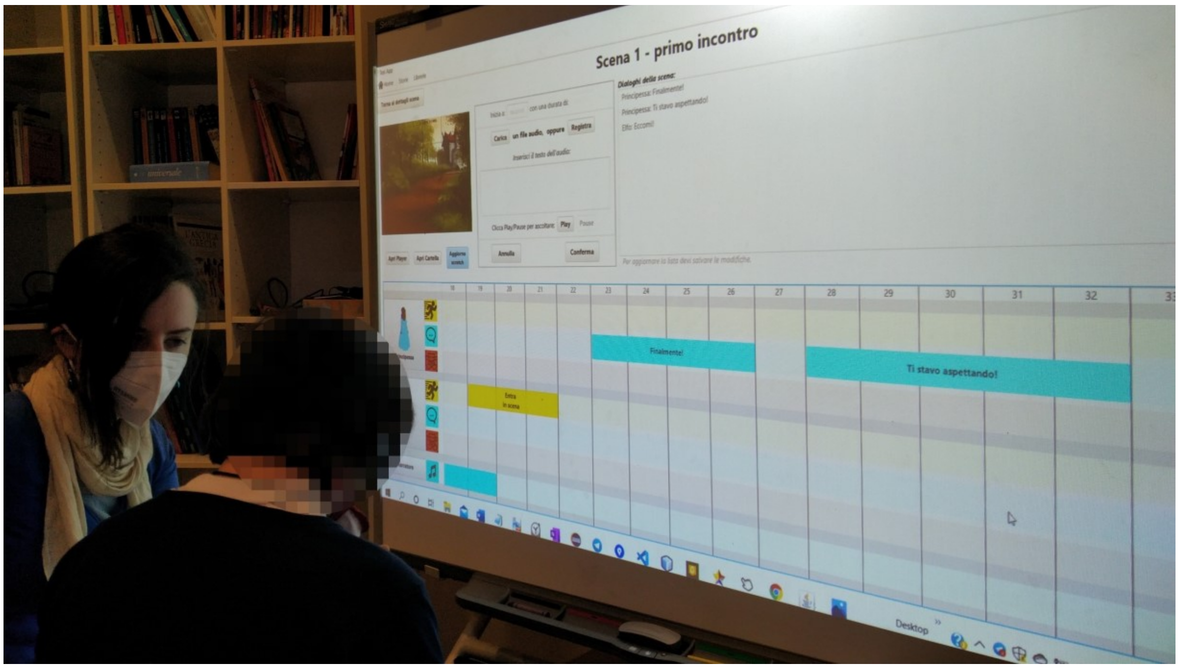


Figure 7.7 - Photo taken during the experiment with subject M

Subject U, assisted by Cristina Bulgheroni. The subject of the third experiment is a 30-years-old boy with severe General Learning Disability. The story made during this test is invented by the subject in all its aspects, settings, characters, and lines.

The problems of concentration and communication are evident. His vocal expression is poor in term of vocal timbre and complex words. During the test, the user appears to be engaged and eager to create a story where the main characters are people he knows and himself. Even if he had his own characters in mind, we did not have time to personalize them on the app, thus we had to use some pre-existing characters with a different physical appearance than what he described. It was not possible to notice a particular evolution within the experiment, as the time available was very limited. We tried to use the guitar both within the story and by recording the boy playing it to insert the sound into the story.

By imagining some characters known to him in the story, he was more immersed in the story and he was capable of expressing himself better. The use of the guitar was a key input as it motivated the boy and made him even more happy about the result obtained with his story.

Narration. N.A.

Lines. There was a constant evolution of the vocal expression and the emotion expression during registration of the lines.

Subject G, assisted by Cristina Bulgheroni. The protagonist of the last experiment is a 20-years-old boy with mental retardation and autistic traits. The story made during this test is invented by the subject, the characters were chosen based on the preferences of G, the settings were already prepared except the last scene, which was entirely built by him.

His vocal expression is poor in terms of vocal timbre and complex words. In the test phase, the user appears active and committed, with a good ability to create and follow the storyline. He needed to divide and shorten sentences into two or three words at most, because it was difficult for him to finish a complete sentence without mumbling half of the words. At the end he looked really satisfied with the story he had created and animated.

The subject showed an evolution during the experiment as his vocal expression significantly improved.

Narration. At the last scene of the story, the emotional identification with the events and the characters was even more evident. The participation in the narration did increase and improve throughout the experiment.

Lines. At the last scene of the story, the line he registered was really good in comparison with those of the first scene. He decided in fact to record the line entirely without interruption and he successfully did, leaving us surprised with both his initiative to do so and his good expression.

Figure 7.8 - Photo taken during the experiment with subject G

7.2.3 Educators comments

In the testing phase, the users appeared active and engaged, with a good ability to interact, speak and listen to the proposed application, considering the limitations imposed by disability. They were all involved in the story, and they all tried to personalize it: M, U, G especially personalized it to a high extent, while A's experiment had a more fixed storyline. Moreover, there was a significant progress in the experiments regarding the vocal expression of the participants as they identified themselves better with the story and they improved in their narration. These results prove the effectiveness of the app with respect to its intended purposes.

Strengths and important features of the App according to educators:

- The easy and fast way of creating the story and reviewing the scenes based on the simple structure of the application allows the user not to get bored and to retain a high level of attention.
- The possibility to decide all the actions during the scene helps the user to observe a continuity in the story and to narrate it in an easy way.
- Reviewing the entire scene while creating it permits the user to stay in the context of the story and to see or listen to the changes in real time (as a result of the new blocks added to the scene).
- The possibility for the users to listen to their voice just recorded while watching the scene evolution, so within a context, helps them and motivates them in improving their vocal expression and their narration of the scenes.

- Inserting in an easy way new characters, objects and backgrounds, permits the participant to completely personalize the story. It is a very important feature for such users (people with intellectual disability) because it is difficult for them to grasp abstract things.

They also suggested some improvements and interesting features to add to the software that we described in the Section 8.2 .

7.3 Testing of application prototype: Amicinsieme

To evaluate the effectiveness of the application, we planned a second experiment session with the association Amicinsieme that lasted two weeks, in which different approach with respect to the one made with Esagramma was adopted.

Amicinsieme is a voluntary association that dedicates all its services to people with disabilities, offering various projects and events that allow the subjects to enjoy their time, learn something new and have fun with volunteers.

In the first Section, the test settings are described, in the second one, the description of the experiment and the results will be provided. Then we will present the feedback of the educator Lorenzo Giuntini[26] who assisted and helped during the sessions.

7.3.1 Test settings

The test was made with the support of a computer and a tv screen. We used the best settings found out in the previous experiment: showing the editor on the computer and the player of the scene/story on the tv screen (see the next Figures for an example of the best setting).

The subjects were: L, E and R. All the three have intellectual disabilities, from mild to severe:

- L: male, 49 years old, mild intellectual disability,
- E: female, 47 years old, mild/moderate intellectual disability,
- R: female, 52 years old, moderate/severe intellectual disability.

The purpose was to evaluate the application from the same three aspects of Esagramma experiment (1,2,3) but also some other more(4,5): (1) the satisfaction of the user with the application, (2) the effectiveness it has in helping the user with intellectual disability improve his/her communication skills, (3) the application's performance levels and error occurrence, (4) the possibility to customize the story as much as you want, (5) the importance and satisfaction to work together (more subjects on the same story).

7.3.2 Description of results

The experiment was structured as follows: (1) choose within which project to make it, (2) decide what to do, (3) create the characters and backgrounds, (4) create the story.

[26] Lorenzo holds a bachelor's degree in psychology and is studying for his master's degree in clinical psychology.

(1) We have chosen to do the testing phase within the project "Fuori tutti" which is a microproject developed in collaboration with the neighbouring kindergarten. The project consists in creating wooden pallets with flowers to decorate the garden where the children play and that is shared with the association.
(2) We decided to create a story to explain the microproject to the children: a simple story related to the garden and the flowers. Thus, the first session was focused on the creation of the idea of the story and the description of the main characters.
(3) Between the first and second session, we created the characters and the backgrounds for the story and added them within the application.
(4) The second and last session was the main one: using the application to create of the story.

In the next paragraphs we will focus on point (4). The Figure 7.9 and Figure 7.11 are two photos taken during the story creation in the second session.

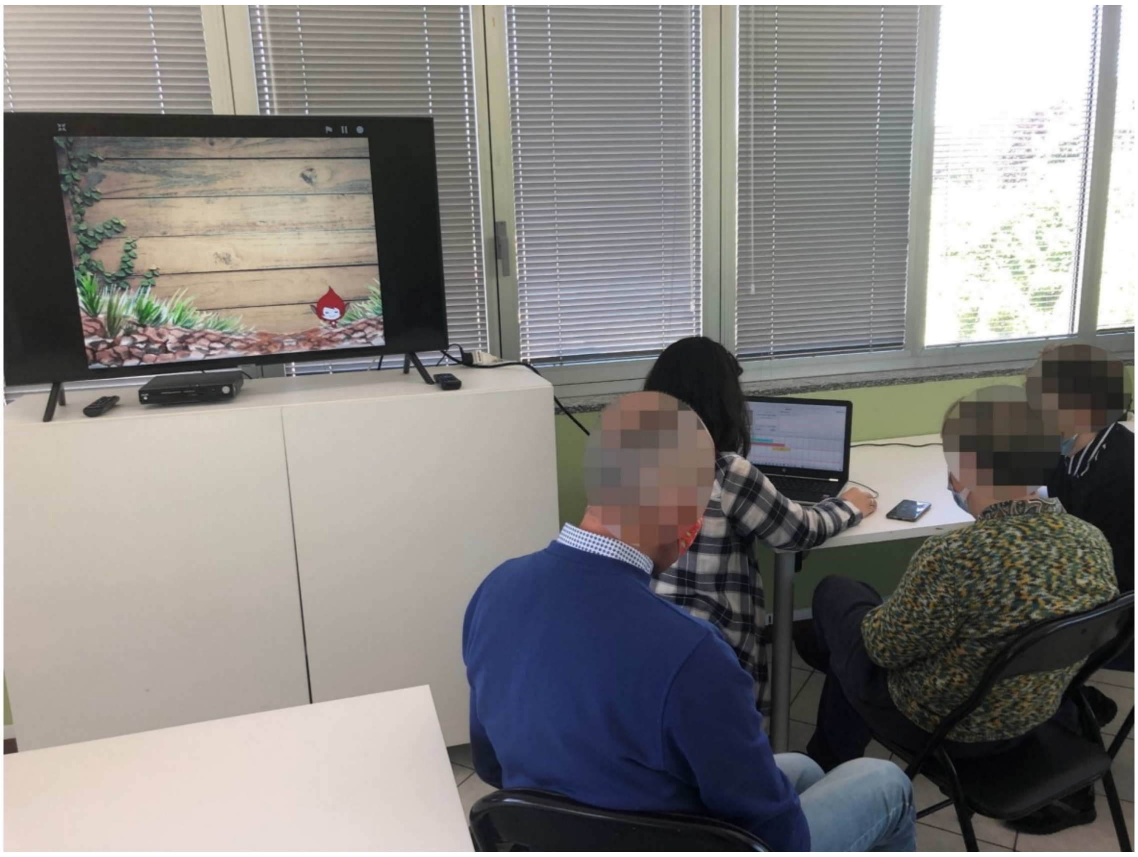


Figure 7.9 - Experiment: Amicinsieme 1

First of all, they choose which character to play. E was the "seed", R the "flower fairy" and L was the "elements of nature" (grain of soil, drop of water and ray of sunshine). The three subjects had to interact with each other through the characters of the story.

Initially, they were not really happy, and some were not even convinced to participate, but, in the end, they were all so enthusiastic that they wanted to do more. Because each of them represented a single character, this helped them to empathize more with it, have more fun and gain more stimuli.

In the last scenes of the story, the lines they registered were significantly better in comparison with those of the first scene and they also express the emotion even better. As the results obtained with **Subject G**, R (the one of the three with more vocal expression difficulties) decided on her own, to record the line entirely without interruption (previously we divided all the lines in smaller parts) and she successfully did it, leaving us surprised with both initiative to do so and good expression.

The sharing time was also successful because they shared with their families the work made together and we will share the story also to the kindergarten children.

As the subjects were excited and there was some time left, we created two new characters and a short new story. The characters of the story had on their face a photo of the subjects (M and E) and a princess dress as the body (see them pointing their character on the screen in Figure 7.10). Seeing themselves dressed as princesses and in new landscapes, made them very happy and involved in the story.

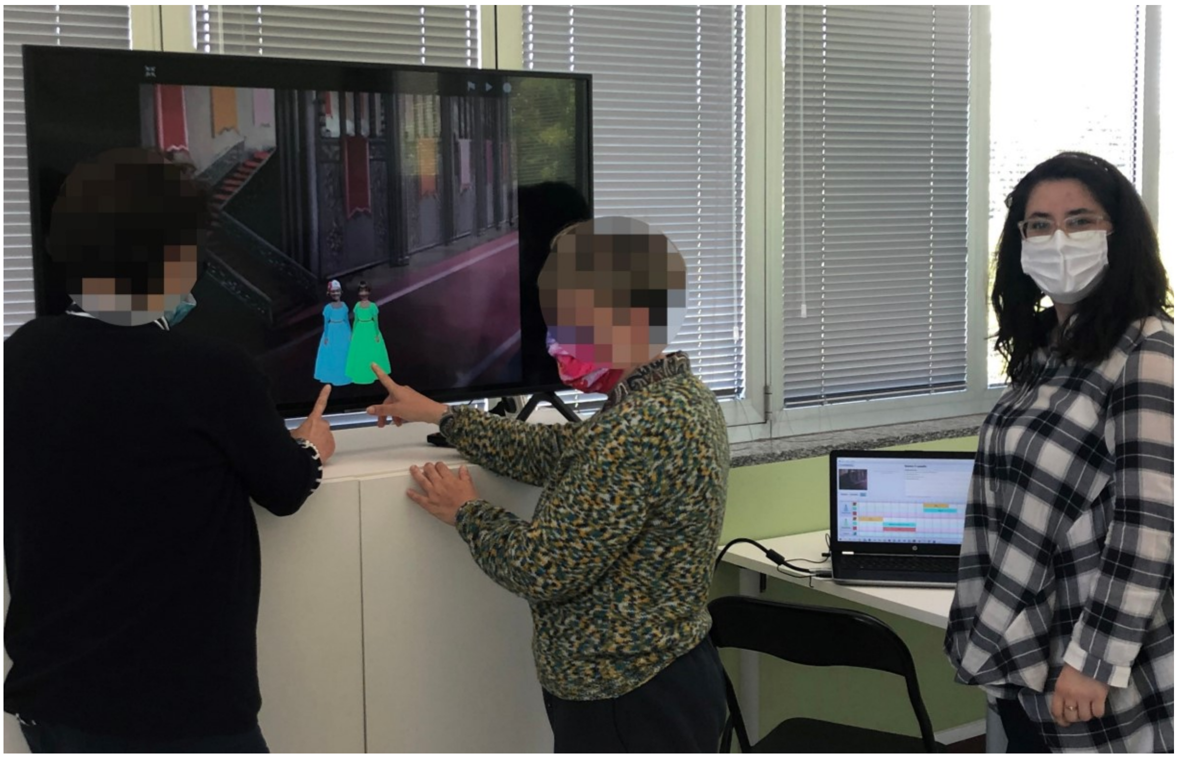


Figure 7.10 - Experiment: Amicinsieme 2

7.3.3 Educator comments

During the tests, we positively noticed how the subjects felt involved with the process of story creation and took an active role in it, and this is very important to ensure they are willing to engage in such kinds of tasks/activities.

One of the positive points about the application is that it creates the opportunity for a cognitive improvement, because the subjects are led to identify with the characters and the hypothetical story, and this is not something they many of them can do frequently in their daily life. Such kind of opportunities allows them to keep their mind active, for instance by following how the story unfolds and understand the importance of the various phases and of the interactions with the others.

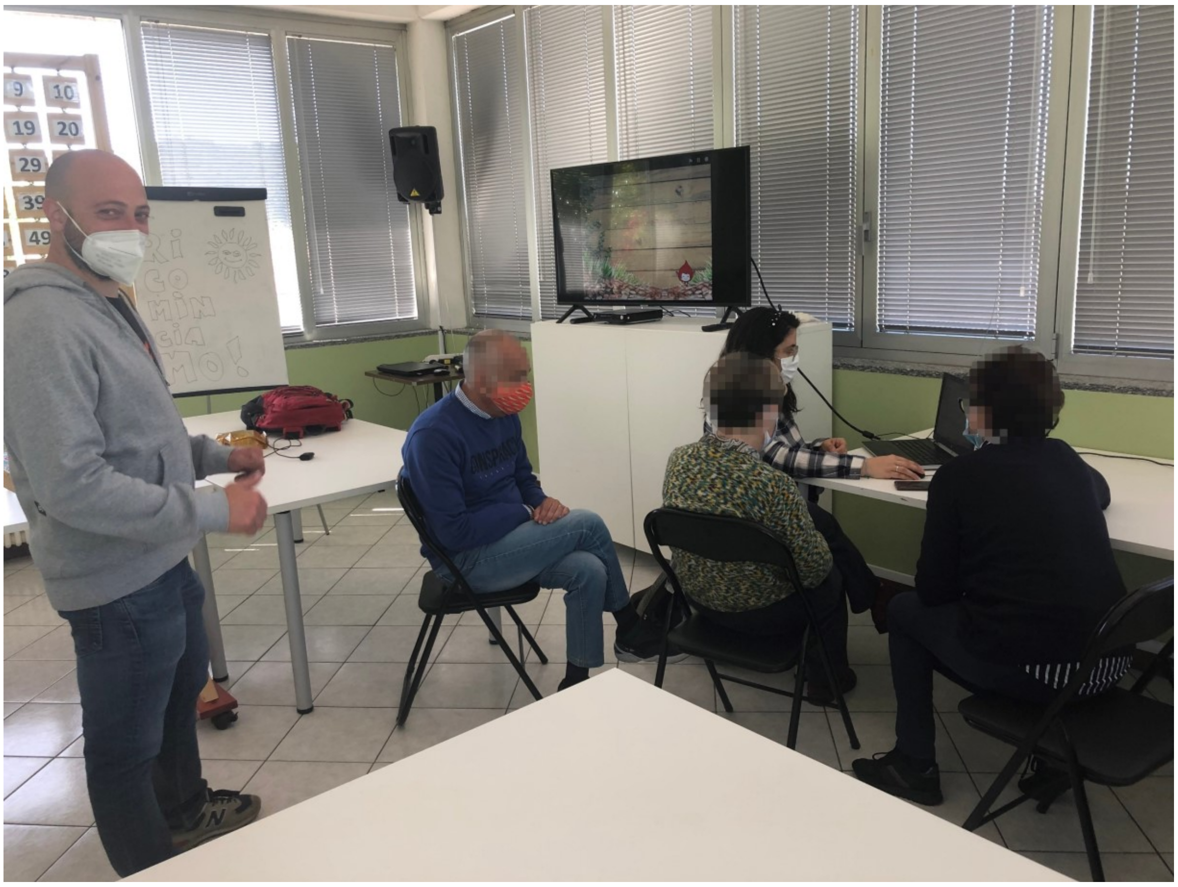


Figure 7.11 - Experiment: Amicinsieme 3

The possibly to use voice to interact with the application allows to obtain improvement also from a speech therapy point of view. Indeed, many people with disabilities can somehow convey what they want to express, but they seldom are able to do so in a precise way and this sometimes can impair their ability to communicate; when using the application, instead, they are pushed to speak in a more precise way to follow the story and this could be of help to improve their ability to express what they want to say even in other contexts.

Lastly, the application also allows to improve the capability to focus in the subjects. Indeed, it is often difficult for people with intellectual disability to remain focused on a task for long. The application, by engaging and requiring them to be always following how the story unfolds so that they can play their part when it is their turn, can be a good way to train them to stay focused for longer in general.

8 Conclusion and Future Works

In this Section we will present the final remarks and the future works of this thesis.

8.1 Conclusion

Starting from the need of providing tools to simplify the process to build an application targeted at the creation of animated stories or sequences of animations we proposed an intermediate language that abstracts the main concepts useful for the creation of the story/animation, being flexible and independent from both the specific application user interface and the execution model.

To validate that our approach can be successfully applied to a real world scenario, we employed it in the design and implementation of an application for storytelling that leverages the intermediate language to generate a Scratch file executable in Scratch player.

Further to this, we have also made use of the increased flexibility provided by the intermediate language during the development phase, when it helped us in developing a few changes to the application, as described in Section 4.7. Finally, in the future we might need to update the app to new Scratch versions, change the target language or add more features, and the IL will make such work easier to perform.

With the testing phase we showed that the application can be useful for the people with intellectual disabilities in the following ways: 1) to improve in communication and verbal expression, 2) to learn to relate more to the world around them, 3) to improve their narration/creativity skills and 4) to learn to recognize and imitate better the emotions.

8.2 Future Works

The proposed Intermediate Language and case study may be evolved in several ways. Some of the most interesting ones are listed below.

8.2.1 Intermediate Language

The semantic of our IL is defined with a Petri net because, as previously discussed, there are many available algorithms for model checking for Petri nets. Therefore, an interesting improvement regarding the translation process would be to design an algorithm for model checking that can verify if the IL description is sound.

A possible extension of the IL features may be to insert a new concept to make the final story interactive: the “choosing point”, where the story will become “interactive” waiting for the user's choice on something and following different path depending on the choice. This may be implemented by creating a new component of the IL: an exclusive gateway taking inspiration from exclusive gateway of BPMN (see Figure 8.1 taken from BPMN official web site)

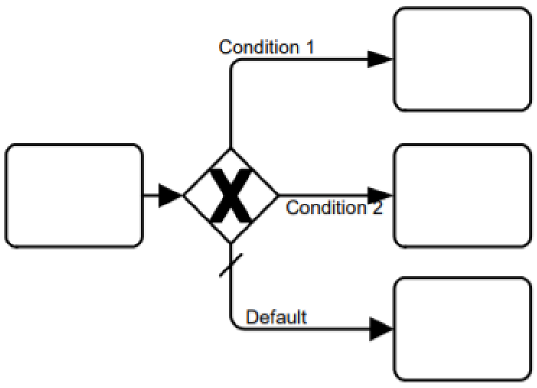


Figure 8.1 - Example of Exclusive Gateway

8.2.2 Application

The following ideas are some improvements of the app that we planned to do in future:

- In the backend of the application there is a concept of “person”, that represents information about the student filled by the educator, and that is also linked to a character that can represent that user as an actor in the story. It would be nice to develop the concept of “person” in the application, so the students could create, with the help of the educator, a character that would act as their avatar, allowing them to directly take part in the stories.
- Another interesting feature that we plan to develop is the “quick editor”. Once a story has been created, the educator might have access to a “quick editor”

where all the voices/sounds related to the characters could be re-recorded, and, at the end, a new story would be saved with all the changes. In this way, should they want to work only on the vocal expression of their students, they could create a default story and then use the "quick editor" to work only on the voice/sound recording.

Other application improvements were suggested from the educators during the testing phase:

- A draft of the scene content for each second inside the editor where to see the changes in real time without re-watching the entire scene.
- The possibility to choose the point x, y clicking on the image with the cartesian plane and not inserting manually the x coordinate and y coordinate of the point.
- Related to the one above, the possibility, after clicking on the point of interest, to see the scene element in the new position.
- A way of choosing the volume for each block that contain audio.

Bibliography

Dann Wanda [et al.] Mediated Transfer: Alice 3 to Java [Journal] // Proceedings of the 43rd ACM Technical Symposium on Computer Science Education. - 2012. - pp. 141–146.

Frädrich Christoph [et al.] Common Bugs in Scratch Programs [Journal] // In Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20). - 2020. - pp. 89–95.

Fraser Neil Ten things we've learned from Blockly [Journal] // 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond). - 2015. - pp. 49-50.

Garzotto Franca and Bordogna Manuel Paper-Based Multimedia Interaction as Learning Tool for Disabled Children [Book Section] // Proceedings of the 9th International Conference on Interaction Design and Children. - 2010.

Maloney John [et al.] The Scratch Programming Language and Environment [Journal] // ACM Transactions on Computing Education. - 2010. - 16. - pp. 16: 1-15.

Merwe A. Thomas, Gerber A. and der A. van A Conceptual Framework of Research on Visual Language Specification Languages [Journal] // 2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD). - 2019. - pp. 1-6.

Ouyang Remco M. Dijkman, Dumas Marlon and Chun Formal Semantics and Analysis of BPMN Process Models using Petri Nets [Journal]. - 2007.

Ouyang Remco M. Dijkman, Dumas Marlon and Chun Semantics and analysis of business process models in BPMN [Journal] // Information and Software Technology. - 2008. - pp. 1281-1294.

Pasternak Erik, Fenichel Rachel and Marshall Andrew N. Tips for creating a block language with blockly [Journal] // 2017 IEEE Blocks and Beyond Workshop (B B). - 2017. - 21-24. - pp. 21-24.

Saridaki Maria and Meimaris Michalis Digital Storytelling for the Empowerment of People with Intellectual Disabilities [Book Section] // Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-Exclusion. - 2018.

Stahlbauer Andreas, Kreis Marvin and Fraser Gordon. Testing scratch programs automatically [Journal] // Proceedings of the 2019 27th ACM Joint Meeting on

European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019). - 2019. - pp. 165–175.

Tahira Shazia AUGMENTATIVE AND ALTERNATIVE COMMUNICATION FOR PEOPLE WITH INTELLECTUAL DISABILITY. - 2020.

Tan Xing, Gu Yilan and Huang Jimmy Xiangji An ontological account of flow-control components in BPMN process models [Journal] // Big Data & Information Analytics. - 2017. - pp. 177-189.

Tanaka Hiroki [**et al.**] Embodied conversational agents for multimodal automated social skills training in people with autism spectrum disorders [Journal] // PLOS ONE. - 2017. - pp. 1-15.

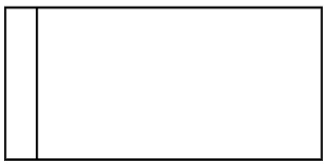
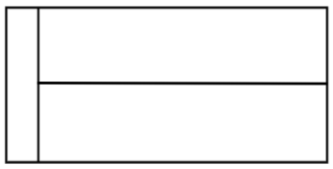
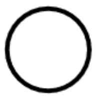
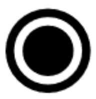
Tiezzi Flavio Corradini [**et al.**] A formal approach to modeling and verification of business process collaborations [Journal] // Science of Computer Programming. - 2018. - pp. 35-70.

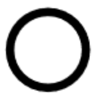
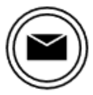
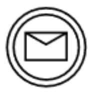
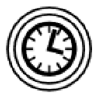
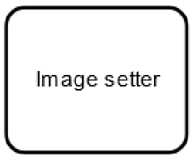
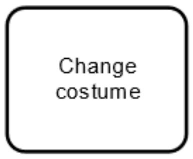
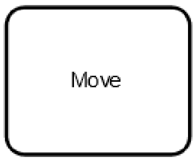
List of Figures

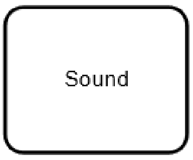
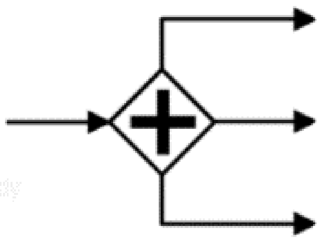
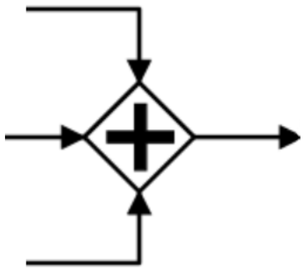
List of Tables

Appendices

Appendix 1: table of the IL components

Graphical element	**Name**	**Meaning**
	**Pool**	Container for the main roles of the project (what we want to translate). There will be three different types of pools: Stage, Sound and Actors.
	**Lane**	The Pool can be divided in lanes. Stage and Sound Pools have only one Lane: Stage lane and Sound Lane respectively, while Actors Pool has a Lane for each participant.
	**Start Event**	The event that represents the start of the project. There will be only one event of this type and it must be placed in the pool *Stage*.
	**End Event**	The event that represents the end of the project. There will be only one event of this type and it must be placed in the pool *Stage*.

	**Middle End Event**	The intermediate end event that is used by the pools that are not stage when they finished all their activities.
	**Send Message Event**	The event that sends a message to another lane.
	**Message received Event**	The event that "wait" to receive a message from another lane or pool
	**Timer Event**	The event that indicates how many seconds the flow needs to wait.
	**Image setter activity**	The activity to set all the initial attributes of the subject.
	**Change costume activity**	The activity to change the look of the subject.
	**Move activity**	The activity to define the movements of the subject. There are six different types of movement: hide, show, glide, move, size, turn.

	**Sound Activity**	The activity to define the sounds of the subject. There are two different types of sounds: text sound and generic sound.
	**Gateway fork**	It divides the flow in three parallel flows
	**Gateway join**	It joins the three parallel flows in a single flow

Appendix 2: Subject A - experiment, report

CORNICE DELLA STORIA: *si propone ad A. di ri-raccontare una delle ultime scene della storia di Red e Toby, protagonista del percorso attuale di Multimedialità.*

I personaggi sono Red e Toby; la scena è predefinita ma è A., guidato, a scegliere quale intervento assegnare a ciascuno.

EVOLUZIONE DELLA CAPACITA' ESPRESSIVA: costante

PARAMETRO	**VALUTAZIONE** (selezionare o scrivere il valore a mano)				
Valutazione globale dell'esercizio di creazione dell'interazione	4	**3**	2	1	0
Comprensione della richiesta	**4**	3	2	1	0
Ricchezza espressiva generale	4	**3**	2	1	0
Coerenza della storia	4	**3**	2	1	0
Capacità di personalizzazione	4	3	**2**	1	0
Vocabolario	Semplice e conosciuto				

Complessità strutturale	Piuttosto semplice					
Emozioni	Paura e riconciliazione					
Aree semantiche	Scappare dal pericolo; amicizia					
EVOLUZIONE FRASI STORIA						
Intelligibilità per la frase migliore	4	**3**	2	1	0	
Intellegibilità media	4	3	**2**	1	0	
Capacità interpretative (esprimere l'emozione o attitudine associata alla frase)	4	**3**	2	1	0	
Numero di ripetizioni massimo	2					
Richiesta prosodica superata	**si**	no				
momenti di crisi **o discontinuità** eventualmente presenti	Necessità di una pausa a metà incontro legata a richieste accattivanti ma faticose.					

Appendix 3: Subject M experiment, report

CORNICE DELLA STORIA: *presente ambientazione predefinita; scelta dei personaggi (principessa e principe). La storia è stata creata da M. sia nella struttura che nei dialoghi.*

EVOLUZIONE DELLA CAPACITA' ESPRESSIVA: si passa dalla fase iniziale in cui è alto il bisogno di sostegno e rassicurazione ad una fase finale in cui cresce l'autonomia nell'esprimersi e la salienza emotiva della storia, da cui la forte immedesimazione.

PARAMETRO	**VALUTAZIONE (selezionare o scrivere il valore a mano)**				
Valutazione globale dell'esercizio di creazione dell'interazione	4	**3**	2	1	0
Comprensione della richiesta	**4**	3	2	1	0
Ricchezza espressiva generale	**4**	3	2	1	0
Coerenza della storia	4	3	**2**	1	0
Capacità di personalizzazione	4	**3**	2	1	0
Vocabolario	Limitato dal deficit comunicativo (legato a decadimento cognitivo)				

Complessità strutturale	Piuttosto semplice				
Emozioni	Imbarazzo e gioia				
Aree semantiche	relazioni affettive (storia d'amore)				
EVOLUZIONE FRASI STORIA					
Intelligibilità per la frase migliore	4	**3**	2	1	0
Intellegibilità media	4	3	2	**1**	0
Capacità interpretative (esprimere l'emozione o attitudine associata alla frase)	**4**	3	2	1	0
Numero di ripetizioni totali	2-3				
Richiesta prosodica superata	**si**		no		
momenti di crisi o discontinuità eventualmente presenti	Alcuni momenti di difficoltà nell'esprimere e condividere i passaggi della storia per come li ha in mente. Altri momenti di difficoltà nell'articolare parole complesse.				

Appendix 4: Subject U experiment, report

CORNICE DELLA STORIA: *si propone ad U. di raccontare storia con i personaggi che vuole della libreria (non si ha il tempo di creare dei nuovi personaggi). La storia è stata creata da U. sia nella struttura che nei dialoghi.*					
EVOLUZIONE DELLA CAPACITA' ESPRESSIVA: costante					
PARAMETRO	**VALUTAZIONE (selezionare o scrivere il valore a mano)**				
Valutazione globale dell'esercizio di creazione dell'interazione	4	**3**	2	1	0
Comprensione della richiesta	**4**	3	2	1	0
Ricchezza espressiva generale	4	3	**2**	1	0
Coerenza della storia	**4**	3	2	1	0
Capacità di personalizzazione	4	**3**	2	1	0
Vocabolario	Semplice e conosciuto				
Complessità strutturale	Piuttosto semplice				

Emozioni	Felicità				
Aree semantiche	Relazioni affettive				
EVOLUZIONE FRASI STORIA					
Intelligibilità per la frase migliore	4	**3**	2	1	0
Intellegibilità media	4	3	**2**	1	0
Capacità interpretative (esprimere l'emozione o attitudine associata alla frase)	4	3	**2**	1	0
Numero di ripetizioni massimo	2				
Richiesta prosodica superata	**Si (in parte)**	no			
momenti di crisi **o discontinuità** eventualmente presenti	/				

Appendix 5: Subject G experiment, report

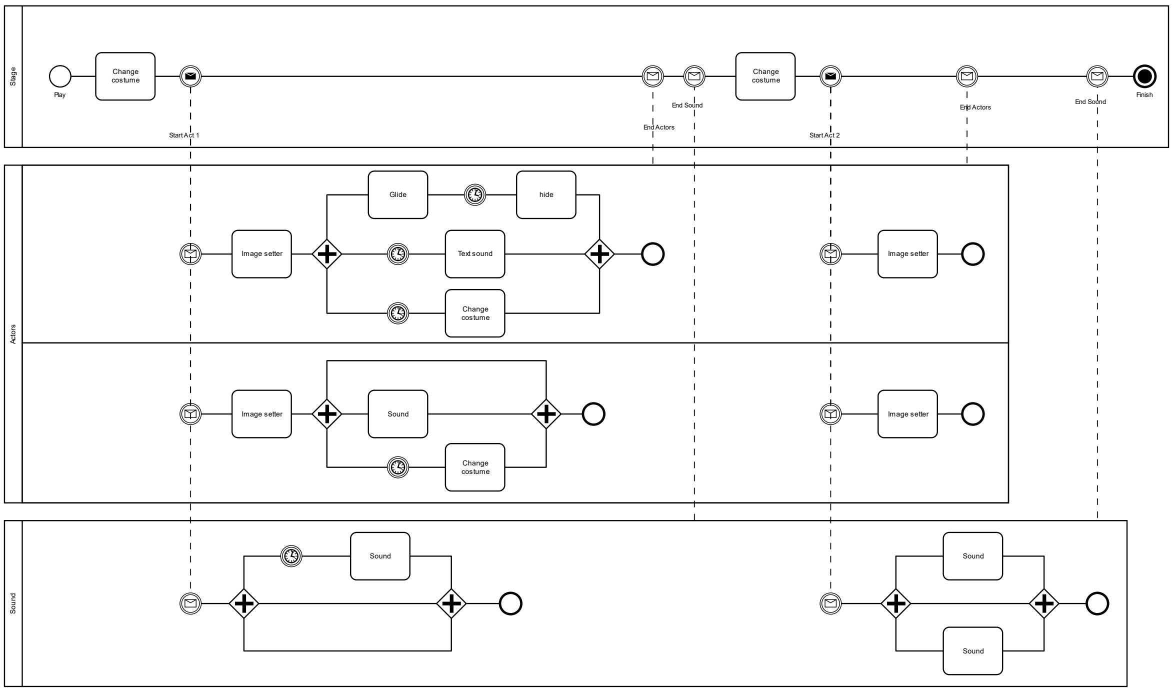
CORNICE DELLA STORIA: *presente ambientazione predefinita (tranne l'ultima scena, la cui ambientazione è stata scelta dal ragazzo); personaggi inseriti in base alle preferenze del ragazzo (principessa, principe, drago). La storia è stata creata da G sia nella struttura che nei dialoghi.*					
EVOLUZIONE DELLA CAPACITA' ESPRESSIVA: si passa dalla fase iniziale in cui è alto il bisogno di sostegno e rassicurazione ad una fase finale in cui cresce l'autonomia nell'esprimersi e la salienza emotiva della storia, da cui la forte immedesimazione.					
PARAMETRO	**VALUTAZIONE (selezionare o scrivere il valore a mano)**				
Valutazione globale dell'esercizio di creazione dell'interazione	4	**3**	2	1	0
Comprensione della richiesta	**4**	3	2	1	0
Ricchezza espressiva generale	4	3	**2**	1	0
Coerenza della storia	**4**	3	2	1	0
Capacità di personalizzazione	4	**3**	2	1	0
Vocabolario	Limitato dal deficit comunicativo				

Complessità strutturale	Piuttosto semplice				
Emozioni	Intraprendenza e gioia				
Aree semantiche	Affrontare il pericolo; relazioni affettive (storia d’amore)				
EVOLUZIONE FRASI STORIA					
Intelligibilità per la frase migliore	**4**	3	2	1	0
Intellegibilità media	4	**3**	2	1	0
Capacità interpretative (esprimere l’emozione o attitudine associata alla frase)	4	3	**2**	1	0
Numero di ripetizioni massimo	2				
Richiesta prosodica superata	**Si (in parte)**	no			
momenti di crisi **o discontinuità** eventualmente presenti	/				

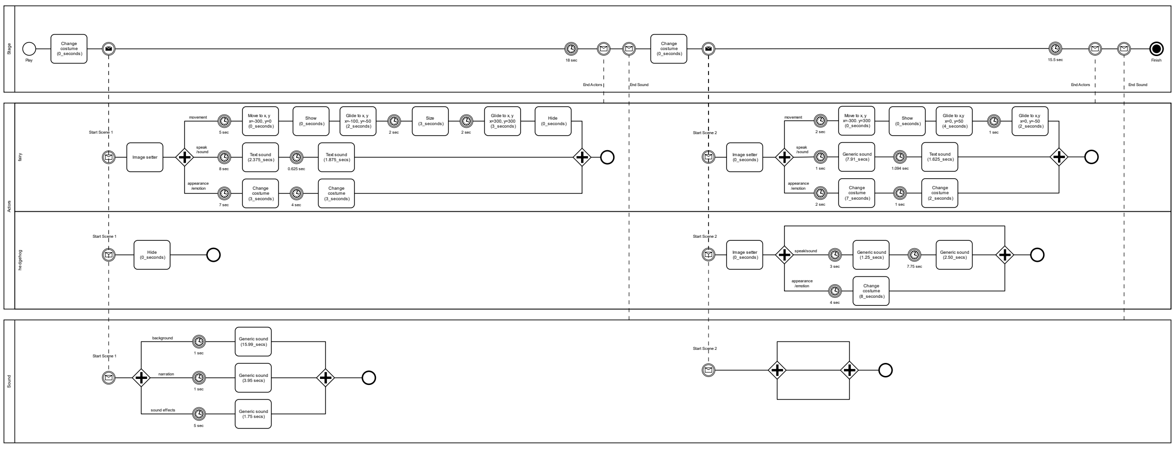
Appendix 6: IL diagrams

Note: The following diagrams are in high quality but not totally correct from the semantic point of view because there are no flow directions due to a Word processor issue. A semantically correct, but lower quality version is available in Section 3.4 (A) and Section 4.6 (B).

(A)



(B)


Stage
Play
Change costume (0_seconds)
18 sec
End Actors
End Sound
Change costume (0_seconds)
15.5 sec
End Actors
End Sound
Finish
Actors
fairy
Start Scene 1
Image setter
movement
5 sec
Move to x, y x=-300, y=0 (0_seconds)
Show (0_seconds)
Glide to x, y x=-100, y=-50 (2_seconds)
2 sec
Size (3_seconds)
2 sec
Glide to x, y x=300, y=300 (3_seconds)
Hide (0_seconds)
speak /sound
8 sec
Text sound (2.375_secs)
0.625 sec
Text sound (1.875_secs)
appearance /emotion
7 sec
Change costume (3_seconds)
4 sec
Change costume (3_seconds)
Start Scene 2
Image setter (0_seconds)
movement
2 sec
Move to x, y x=-300. y=300 (0_seconds)
Show (0_seconds)
Glide to x,y x=0, y=50 (4_seconds)
1 sec
Glide to x,y x=0, y=-50 (2_seconds)
speak /sound
1 sec
Generic sound (7.91_secs)
1.094 sec
Text sound (1.625_secs)
appearance /emotion
2 sec
Change costume (7_seconds)
1 sec
Change costume (2_seconds)
hedgehog
Start Scene 1
Hide (0_seconds)
Start Scene 2
Image setter (0_seconds)
speak/sound
3 sec
Generic sound (1.25_secs)
7.75 sec
Generic sound (2.50_secs)
appearance /emotion
4 sec
Change costume (8_seconds)
Sound
Start Scene 1
background
1 sec
Generic sound (15.99_secs)
narration
1 sec
Generic sound (3.95 secs)
sound effects
5 sec
Generic sound (1.75 secs)
Start Scene 2