ERPL: DSL for escape rooms

Tiago Luís Dias da Silva
Departamento de Informática, Universidade do Minho, Braga, Portugal,

José João Almeida
Algoritmi, LASI, Departamento de Informática, Universidade do Minho, Braga, Portugal

Abstract

This article delves into the development of ERPL, a Domain-Specific Language tailored for virtual escape rooms. It explores the mechanics, technology, and architecture used in creating immersive and engaging virtual escape room experiences. Case studies demonstrate the practical application of ERPL in designing virtual escape rooms, showcasing its flexibility and effectiveness in meeting the demands of escape room creators. In conclusion, we summarize the results obtained and discuss potential future directions for the development of ERPL.

2012 ACM Subject Classification Domain specific language

Keywords and phrases Escape rooms, Domain-Specific Language, Game development

Digital Object Identifier 10.4230/OASIcs.SLATE.2024.

Acknowledgements This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

1.1 Motivation

Escape rooms are immersive and challenging games in which participants find themselves locked in a themed room, facing a series of puzzles and riddles to discover an exit route within a set time. This form of entertainment has grown in popularity not only in leisure but also in areas such as education [1, 2, 3] and tourism [4].

In the educational field, escape rooms have been embraced as powerful tools for gamification and game-based learning. Their effectiveness transcends different levels and disciplines, providing engaging and memorable learning experiences.

With the advancement of technology, escape rooms have evolved into virtual versions, further expanding the possibilities of this type of entertainment.

This article proposes the creation of a Domain-Specific Language (DSL) aimed at the development of virtual escape rooms. One of the motivations behind this initiative is to enable educators to conceive and implement escape rooms simply and intuitively, without relying solely on advanced programming skills. Additionally, we want the ERPL (Escape Room Programming Language) to be a language that empowers programmers who want to undertake more ambitious and complex projects.

This approach aims to make the creation of escape rooms more accessible and inclusive, broadening the possibilities of using this format of entertainment and learning in various areas and contexts.

1.2 Goals

The ERPL has been designed with a set of clear objectives in mind, aiming to meet the needs of escape room creators and facilitate the development process. These objectives include:

- **Facilitating the creation of escape rooms:** The DSL aims to provide escape room creators with the ability to develop scenarios, puzzles, and interactions quickly and efficiently, without requiring advanced programming knowledge. This will be achieved through the implementation of specific abstractions for common elements found in escape rooms, such as scenes, objects, and puzzles.
- **Promoting reusability and modularity:** The ERPL aims to facilitate the reuse of escape room components and promote modularity in design, allowing creators to share and combine different elements flexibly. To achieve this, clear interfaces and abstractions will be defined to separate the various parts of the game, making composition and adaptation easier.
- **Ensuring scalability and extensibility:** The language will be designed with scalability and extensibility in mind, allowing escape room creators to add new features and functionalities as their needs evolve. This will be enabled through a flexible and modular architecture that supports the addition of new elements and behaviors without compromising the integrity of the language core.

These fundamental objectives guide the development of ERPL, ensuring that it meets the demands of escape room creators effectively and flexibly [6, 8].

1.3 Article structure

This article is structured as follows:

- **Introduction**: This section provides the context and motivation behind the development of ERPL, as well as its objectives and the structure of the article.
- **State of the Art**: In this section, we provide an overview of the current state of escape rooms, including their different typologies and related work. We also discuss the technologies used in this domain.
- **ERPL Specification**: Here we describe the architecture, model, and syntax of ERPL, detailing how the language is designed to facilitate the creation of escape rooms.
- **Case Studies**: We present case studies demonstrating the practical application of ERPL in the creation of virtual escape rooms.
- **Conclusion**: We conclude the article by summarizing the results obtained and discussing possible future directions for the development of ERPL.

This structure is designed to provide a comprehensive overview of the article's content, allowing readers to easily understand the key aspects of ERPL and its application in creating escape rooms.

2 State of the art

2.1 Escape Room Typologies

Following the article written by Nicholson [5] on the origin and evolution of escape rooms, we can see that escape rooms encompass a variety of experiences that can be adapted to a virtual approach through ERPL. Below, we highlight the main aspects to consider:

2.1.1 Themes

Escape rooms can offer a wide range of themes, such as medieval, futuristic, and horror. In the virtual version, this diversity is explored through images and audio, allowing for customization and expansion of thematic options. ERPL should facilitate the import of assets chosen by the user and provide specialized templates in different themes.

2.1.2 Formats

To simplify the scope of ERPL, we will focus solely on virtual escape rooms in Two Dimensions (2D), avoiding the complexity of Three Dimensions (3D) and Virtual Reality (VR) versions.

2.1.3 Mechanics

The ERPL should cover several common mechanics found in different virtual escape room games, including:

- **Point-and-Click:** Simple interactions through clicks to explore and interact with game objects. (Game example: The Crimson Room[1])
- **Drag-and-Drop:** Tactile manipulation of objects to solve puzzles. (Game example: The Room[2])
- **Text Interactions:** Dialogues and descriptions for inserting commands and receiving important information. (Game example: Zork[3])
- **Logical Flow:** Linear progression in gameplay, where solving puzzles leads to other challenges. (Game example: Myst[4])
- **Multiple Paths:** Choices that impact the unfolding of the story, providing different outcomes. (Game example: Her Story[5])
- **Cooperative Elements:** Features for collaboration among players, even at a distance. (Game example: Keep Talking and Nobody Explodes[6])

These mechanics are fundamental to creating an immersive and engaging experience in virtual escape rooms developed in ERPL.

2.2 Technology Used

To develop ERPL and its associated infrastructure, the following technologies were used:

The ERPL compiler was implemented in Python, utilizing the Lark library. Lark[7] is a library for analyzing and manipulating formal grammars in Python. It provides a convenient way to define language grammars and parse strings according to those grammars. With Lark, it was possible to create an efficient and flexible parser for the ERPL language, facilitating the interpretation and compilation of escape room scripts.

The engine responsible for making the escape rooms playable was developed in Python, using the Pygame[8] library. Pygame is an open-source, cross-platform library designed to facilitate game development in Python. It offers features for handling graphics, sound, user input, and more, making it a suitable choice for implementing the escape room engine.

[1] `https://www.crazygames.com/game/crimson-room`
[2] `https://store.steampowered.com/app/288160/The_Room/`
[3] `https://store.steampowered.com/app/570580/Zork_Anthology/`
[4] `https://en.wikipedia.org/wiki/Myst`
[5] `https://store.steampowered.com/app/368370/Her_Story/`
[6] `https://keeptalkinggame.com`
[7] `https://lark-parser.readthedocs.io/en/latest/index.html`
[8] `https://www.pygame.org`

3 ERPL Specification

3.1 Architecture

The system architecture consists of a compiler that takes as its main source a script written in our language (file .er) along with multimedia files, which can be images, music, or objects from a language assets library. The compiler generates a model of the escape room in a JSON file, which will later be read by an engine to create a playable application of the escape room. Currently, we have developed an engine in Python using the Pygame library, but in the future we plan to create an engine to allow gameplay on the web, in order to make it easier for users to use.

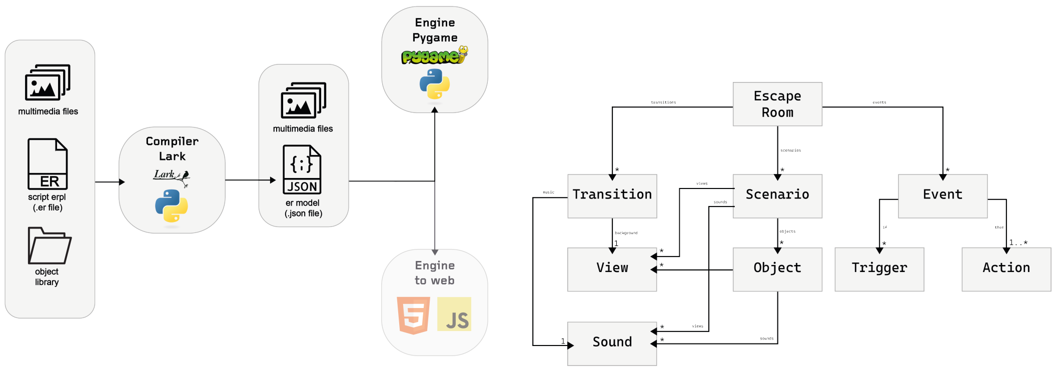


Figure 1 Architecture and model

3.2 Model

- **Escape Room:** The cornerstone of our model. It consists of three fundamental components: scenarios, transitions, and events. It also has a title and a size.
- **Scenarios:** These constitute the core environments within the escape room. Each scenario encompasses various objects, visual representations (views), and audio elements (sounds). The size associated with scenario views is always the size defined in the escape room, and their position is set to (0,0).
- **Objects:** Objects are associated with a scenario and can have multiple views and sounds. Additionally, they can be associated with a position and size, which will be applied to views of the object that do not have these fields defined.
- **Transitions:** Responsible for guiding the narrative flow of the escape room experience. Transitions include a background visual, music, and narrative text. They also define the subsequent step in the sequence, whether it is another transition or a scenario.
- **Events:** These serve as trigger-action mechanisms within the escape room. Events are activated by specific triggers, such as a click on an object. Once triggered, they initiate actions such as changing an object's view or transitioning to a different scenario. We will delve into further detail on the possible triggers and actions in the syntax subsection.
- **Challenges:** Challenges are a type of special action that can be invoked in events. These represent unique interactions with the player, where success or failure can lead to specific outcomes. In our language, there are various types of challenges, each with its own mechanics and consequences. We will delve into further detail on the different types of challenges in the following subsection dedicated to syntax.

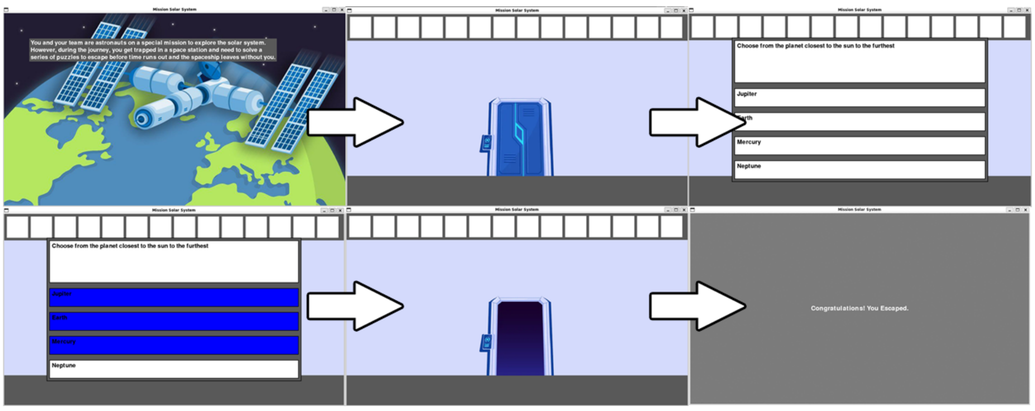
- **View:** A view provides visual support and can be either a static image or multiple images to form an animation. It also has a size and a position.
- **Sound:** A sound serves as our auditory support and is associated with a sound source.
- **Inventory:** The inventory is a concept that defines the objects that the player possesses and can use in the scene or on objects within it.

3.3 Syntax

Given the limited space we have, we've decided to showcase syntax with a simple example that demonstrates the core of our language. Here is the definition of an escape room that consists of only one scenario with one object, a starting transition, and a few events for interaction.

Listing 1 Escape Room

```
EscapeRoom( #Escape Definition
    title="Mission Solar System",size=[1280,720],
    scenarios=[station],
    events=[try_open,event_open,show_error,exit],
    transitions=[intro],start=intro)
intro = Transition( #Intro Transition definition
        background = View.Static.station(image="station.png"),
        music = Sound.music(source="intro.mp3"),
        story = "You and your team are astronauts on a special mis...")
#Object Door definition
sound_opening = Sound(source="door-opening.mp3")
door_closed = View.Static(image="door_closed.png")
door_opened = View.Animated(
    images=["door_semi_closed.png","door_open.png"],
    repetitions=1, time_sprite=3)
door = Object(initial_view = door_closed,
              views = [door_closed,door_opened],
              position = (497,310), size=[286,300],
              sounds = [sound_opening])
#Station Scenario definition
background = View.Static(image="room.png")
station = Scenario( initial_view = background,
                    views = [background],
                    objects = [door],sounds = [])
#Events Definitions
event_open = Event(
    then = door change to door_opened and play sound_opening of door)
show_error = Event(then = show message "Wrong!" in (497,310))
question_planets = Challenge.Sequence(
                question= "Choose from the planet clos...",
                sequence= ["Mercury","Earth","Jupiter","Neptune"],
                sucess= event_open,fail=show_error)
try_open = Event(if = click door and door is door_closed,
                then = start challenge question_planets)
exit = Event(if = click door and door is door_opened, then = end of game)
```


You and your team are astronauts on a special mission to explore the solar system. However, during the journey, you get trapped in a space station and need to solve a series of puzzles to escape before time runs out and the spaceship leaves without you.

Choose from the planet closest to the sun to the furthest

Choose from the planet closest to the sun to the furthest

Congratulations! You Escaped.

Figure 2 Example

Next, we present the various challenges that can be created using our language, all of which have the parameters 'success' and 'fail' associated with events and follow the following syntax:

Listing 2 Challenges

```
Challenge.ChallengeType(parameter=parameter_value,...,
                        sucess=Event_id,fail=Event_id)
```

Table 1 Syntax, Description, and Parameters of Challenges

ChallengeType	Description	Parameters
Question	Asks a question and expects a specific answer.	`question=Text`, `answer=Text`
Motion	Requires a specific motion or action to be performed.	`motion_object=Object_id`, `trigger_object=Object_id`
Multiple Choice	Presents a multiple-choice question with options.	`question=Text`, `choices=[Text]`, `answer=Text`
Connection	Requires matching items from two lists.	`question=Text`, `list1=[Text]`, `list2=[Text]`
Sequence	Asks for items to be arranged in a specific sequence.	`question=Text`, `sequence=[Text]`
Puzzle	Displays an image puzzle to solve.	`image=Text`
Slide Puzzle	Presents a slide puzzle to solve.	`image=Text`

Here is the syntax for all possible triggers and actions in the language.

Table 2 Triggers and Actions

Triggers
click Object_id
click not Object_id
Object_id is View_id
Event_id already happened
Object_id is in use
Number seconds have already passed

Actions
Object_id change to View_id
Object_id goes to inventory
end of game
show message Text in Position
Object_id change size to Size
Object_id move to Position
change to Scenario_id
Object_id is removed
play Sound_id of Object_id|Scenario_id
start challenge Challenge_id
transition Transition_id

In the language, as seen in the architecture, it's possible to import objects from a library. These must be imported after defining the escape room and before any other assignments.

Listing 3 Imports Example

```
import Object.door
door1 = Object.door
door2 = Object.door(position = (100,100))
#It is possible to change some parameters of an imported object.
```

Finally, in the language, it is also possible to write using Python code. For this, at the end of the script, a block can be dedicated to running Python code, and then variables and functions can be used in assignments as seen in this syntax example. Types are compared and validated during compilation. Below is a simple example where Python is used to define the list of images for animating a door.

Listing 4 Python Block

```
door_animated_example1 = Python.door
door_animated_example2 = Python.my_glob("door")
__Python__ #This is python code
import glob
door = glob.glob("../assets/images/door/*.png")
def my_glob(object):
    return glob.glob(f"../assets/images/{object}/*.png")
```

4 Conclusion

We introduced ERPL (Escape Room Programming Language), a language designed to streamline the creation of virtual escape rooms. Throughout the article, we explored its architecture, syntax, and use cases, highlighting its effectiveness and flexibility in crafting immersive experiences. Looking ahead, there are several exciting avenues for future development of ERPL:

- Enhance the engine to improve performance and add new features.

- Introduce a variety of challenges to enrich player experiences.
- Develop a web engine and block-based language to expand the reach of virtual escape rooms. We have already started with the development of both the engine, which is being done with the help of the p5[9] library for javascript, and the block-based language using the Google Blockly[10] library [7].
- Create usable templates and scripts to streamline development processes.
- Implement a graphical interface that integrates with the language to simplify escape room creation.

With these future developments, we anticipate ERPL becoming an even more powerful and accessible tool for designers and escape room enthusiasts, driving innovation and creativity in this exciting and expanding field. At this moment all development can be consulted and installed on the pypi project's [11].

References

1 Aisyah Saad Abdul Rahim, Mohd Shahezwan Abd Wahab, Aida Azlina Ali, and Nur Hafzan Md. Hanafiah. Educational escape rooms in pharmacy education: A narrative review. *Pharmacy Education*, 22(1):p. 540–557, 06 2022. `doi:10.46542/pe.2022.221.540557`.

2 Samantha Clarke, Daryl Peel, Sylvester Arnab, Luca Morini, Helen Keegan, and Oliver Wood. Escaped: A framework for creating educational escape rooms and interactive games to for higher/further education. *International Journal of Serious Games*, 4, 09 2017. `doi:10.17083/ijsg.v4i3.180`.

3 Mário Cruz. 'escapando de la clase tradicional': the escape rooms methodology within the spanish as foreign language classroom. *Revista Lusófona de Educação*, 46:117–137, 12 2019. `doi:10.24140/issn.1645-7250.rle46.08`.

4 Arsenio Villar Lama. Millennial leisure and tourism: The rise of escape rooms. *Cuadernos de Turismo*, 41:743–746, 2018.

5 Scott Nicholson. Peeking behind the locked door: A survey of escape room facilities. White Paper, 2015. Available at `http://scottnicholson.com/pubs/erfacwhite.pdf`.

6 Nuno Oliveira, Maria João Pereira, Pedro Rangel Henriques, and Daniela Cruz. Domain specific languages: a theoretical survey. In *INFORUM'09 Simpósio de Informática*. Faculdade de Ciências da Universidade de Lisboa, 2009. URL: `http://hdl.handle.net/10198/1192`.

7 Erik Pasternak, Rachel Fenichel, and Andrew N. Marshall. Tips for creating a block language with blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 21–24, 2017. `doi:10.1109/BLOCKS.2017.8120404`.

8 Markus Voelter. From general purpose languages to dsls. In Markus Voelter, editor, *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*, chapter 2.2, pages 25–31. CreateSpace Independent Publishing Platform, 2013.

[9] `https://p5js.org`
[10] `https://developers.google.com/blockly?hl=pt-br`
[11] `https://pypi.org/project/erpl/`