RPG Maker Puzzle Archetypes: Analysis, Evaluation, Refactoring and Extension

Joimar Brito Gonçalves Filho, João Gabriel Lima Moraes, Victor Travassos Sarinho
Universidade Estadual de Feira de Santana (UEFS)
Laboratório de Entretenimento Digital Aplicado (LEnDA)
Feira de Santana, Bahia, Brazil
joimarbgf@gmail.com, joaofeirense@gmail.com, vsarinho@uefs.br

***Abstract*—The presence of puzzles in RPGs is very common in the history of digital games, being a foundation for the development of different successful games in the industry. This paper presents a study of RPG puzzle archetypes provided by the RPG Maker MV developer community. For this, the identification of puzzle types, the selection of puzzle-making techniques, the implementation and refactoring of puzzle prototypes and the extension of evaluated puzzle archetypes were performed.**

***Keywords*-rpg maker mv; rpg puzzles; puzzle archetypes;**

I. Introduction

The genre concept has an important influence on digital games, taking into account game goals and objectives, the gameplay nature, and even the game characters themselves [1]. Puzzle games, for example, represent a game category that does not have a formal definition, but have a set of characteristics and limitations that usually involve shapes, colors or symbols that must be arranged to produce a certain pattern [2]. In the same way, the Role Playing Games (RPGs) category puts the player into a fictional scenario, assuming different responsibilities based on the established narrative and defined roles in the game [3].

The presence of puzzles in RPGs is very common, working as a model for the development of several blockbuster games around the world (*The Legend of Zelda* and Final Fantasy, for example). This combination amplifies the complexity of a game, contributing to the player immersion who needs to look for a solution in every mystery proposed by the fictional game world.

Regarding the development of RPGs, several tools have been built along the years to achieve a quick production cycle. As an example, the RPG Maker is a framework that has become one of the most popular game engines for RPG development [4]. However, the native RPG Maker resources to create puzzles are rather limited, demanding the use of more complex scripts to develop better elaborated puzzles [5]. The lack of a formal documentation to teach how to create determined puzzle types represents a current development challenge in the RPG Maker community, creating a dependency with development forums to find compatible puzzle models to build the desired application.

This article presents the analysis of some puzzle models capable of being applied in the development of distinct RPGs. The selected development platform was the RPG Maker MV and the following studies were conducted: 1) Identification of preconceived puzzle types for the framework; 2) Selection of puzzle-making techniques released by the development community; 3) Implementation of puzzle prototypes based on selected techniques, including feasibility verification and developed refactoring; and the 4) Extension of the evaluated puzzles through the addition of personalized code and community created plugins.

II. Methodology and Results

A. Collection of Puzzles

There are different types of mechanics that guide the creation of digital RPGs [6]. However, there is no consensus over the categorization for puzzles in RPGs, requiring an extra effort to fit them in specific categories. In this sense, the first stage of this work is to describe the types of puzzles preconceived and indicated by the RPGMaker tool, such as:

- *Sequence*: Requires a sequence of actions in the correct order, such as “pulling lever A and then B” or “lighting lamp 1 and then lamp 2”. Such puzzles usually become memory games, where the game shows a correct order and requires the player to repeat it.
- *Mechanism*: Set of keys that require a switch to be activated to open a door. Sometimes involves a pressure point, where an object must be placed over the point in question to keep it active. Unlike the Sequence puzzles, the change of one switch value in a Mechanism puzzle can modify the state of the others.

As second stage of this work, a research was made in RPGMaker development communities looking for Sequence and Mechanism puzzles represented as puzzle archetypes that could be used by this tool. As a result, 3 common puzzle archetypes were found:

- *Password*: Sharing Sequence and Mechanism characteristics, this archetype uses a finite number of attributes that have to be inserted. If the position of the inputs are correct, the password is valid. Numbers, words, colours or any other information that can be transcribed as a logic value can be used as a password for this archetype.

- *Sliding Block*: Represents a hybrid between Sequence and Mechanism puzzles, once it is based on interaction with objects and positions in a scenario. For this archetype, the order in which the player actions are performed influences the end result to solve the puzzle.
- *Six Lights*: Represents a Mechanism puzzle with an interdependence between switches. Basically this type of puzzle proposes to the player to turn all switches from state A to state B, but for each switch, once its current state changes, the state of other nearby switches will also change.

B. Evaluating Puzzle Archetypes

To verify and validate identified puzzle archetypes, some refactored prototypes were developed. They are based on the original code of each previously mentioned archetype, as the original versions have a poor performance or design limitation in some aspects.

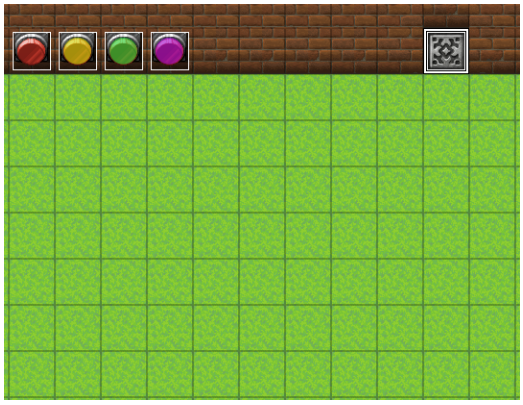

Figure 1. *Password* puzzle example.

1) Password: As illustrated by this archetype example (Figure 1), there is a four buttons panel to be used by the player, where each button will change the color when pressed. If the buttons are pressed in the correct order, the door in the scene will open.

Similar algorithm and performance were identified in Password codes during the research, making unnecessary to choose an specific code to evaluate this puzzle archetype. In this sense, the selected code from the development community (Figure 2) applies a password logic by the interaction between different buttons, and each one is capable of assuming a true or false logic value.

```
◆Play SE : Switch3 (90, 100, 0)
◆Control Self Switch : A = ON
◆Control Self Switch : B = OFF
◆Control Variables : #0001 Door Switch Puzzle += 30
```

Figure 2. Code of a button used by the password puzzle.

```
◆If : paswd1 = 3
  ◆If : paswd2 = 2
    ◆If : paswd3 = 3
      ◆If : paswd4 = 0
        ◆Set Movement Route : This Event (Wait)
        :                   : ◇Turn Left
        :                   : ◇Turn Right
        :                   : ◇Turn Up
        ◆Text : None, Window, Bottom
        :     : a porta está aberta
        ◆Control Self Switch : A = ON
```

Figure 3. Refactored code of the password puzzle.

The code makes each switch command to increment the value of a specific integer variable. Each button has a specific weight value that is summed across themselves. If the result is a predetermined correct value, the puzzle is solved. It is noticeable that this strategy only works if the buttons can assume boolean values, requiring an algorithm change if the buttons are designed to assume a range of values.

In the refactored code every button is associated to a specific variable ("**passwd**" followed by a number) and color (red, green, yellow and purple), where the colors are linked to an integer value from these variables (Figure 3). Once the variable values are according to what the conditional structures demands, the system will see the password as correct.

2) Sliding Block: As illustrated in Figure 4, the scenario is showing some barrels that represent obstacles to the stones. They stop the sliding motion of the stones, keeping them standing in a useful position for puzzle dynamics. The computational logic, which recognizes the correct position of each stone is responsible to solve the proposed puzzle.

Figure 4. *Sliding Block* puzzle example.

In the refactored code, it was changed from the original to minimize the memory access and reduce the effects of excessive logic checks (Figure 5).

The system monitor controls all moving parts of the puzzle using "**boulder**" named variables. If the "stone 1", associated to the "**boulder1x**" and "**boulder1y**" variables, is in the correct position, which is indicated by the "**posx**" and "**posy**" variables, the system recognizes that the "stone 1" is in the correct position (Figure 6). To solve the puzzle, all four stones need to be in their respective correct position .

3) Six Lights: As illustrated in Figure 7, the scenario represents six different lights that change their own color when the player interacts with them. The puzzle objective is to inverse all crystal light values from the original one, where any interaction with one light influences other nearby lights.

Each crystal is represented by the respective variable "**c...**" (Figure 8). When an interaction occurs, the crystal inverts the current value and the values of others crystals

```
◆If : Script : (getEventX(3) == getEventX(1) || getEventX(3) ==
  ◆If : Script : (getEventX(4) == getEventX(1) || getEventX(4)
    ◆Show Animation : Player, Fog
    ◆Text : None, Window, Bottom
    :     : Puzzle Complete!
    ◆Play ME : Victory2 (90, 100, 0)
    ◆
  : End
  ◆
: End
◆
```

Figure 5. Partial description of the *Sliding Blocks* original code.

```
◆Control Variables : #0014 boulder1x = Map X of boulder_1
◆Control Variables : #0015 boulder1y = Map Y of boulder_1
◆Control Variables : #0016 boulder2x = Map X of
◆Control Variables : #0017 boulder2y = Map Y of
◆Control Variables : #0018 boulder3x = Map X of boulder_3
◆Control Variables : #0019 boulder3y = Map Y of boulder_3
◆Control Variables : #0020 boulder4x = Map X of boulder_4
◆Control Variables : #0003 boulder4y = Map Y of boulder_4
◆Control Variables : #0001 pos_x = Map X of This Event
◆Control Variables : #0002 pos_y = Map Y of This Event
◆If : boulder1x = pos_x
  ◆If : boulder1y = pos_y
    ◆Play SE : Magic1 (90, 100, 0)
    ◆Control Switches : #0001 c1 = ON
    ◆
  : End
  ◆
: End
◆
```

Figure 6. Refactored *Sliding Blocks* code.

Figure 7. *Six Lights* puzzle example.

```
◆Control Switches : #0001 c1 = ON
◆If : c2 is OFF
  ◆Control Switches : #0002 c2 = ON
  ◆
: Else
  ◆Control Switches : #0002 c2 = OFF
  ◆
: End
◆
```

Figure 8. Refactored *Six Lights* code.

```
/* NEW FUNCTIONS */

function toggleEvLo(id) {
    $gameMap._events[id].lo=!$gameMap._events[id].lo;
}

function getEvLo(id) {
    return $gameMap._events[id].lo;
}

(function() {
    var tmp = Game_CharacterBase.prototype.initMembers;
    Game_CharacterBase.prototype.initMembers = function() {
        tmp.call(this);
        this.lo=false;
    };
})();

/*
Conditional Branch:
getEvLo(2)===true && getEvLo(3)===true && getEvLo(4)===true &&

*/
```

Figure 9. Partial description of the original Six Lights code.

next. The comparative study between the initial version and the refactored one reveals that the original implementation performs an extra memory access (Figure 9), making an unnecessary checking in each variable associated to the crystal.

Figure 10. Light detection mechanics provided by a RPG Maker plugin.

```
◆Control Variables : #0001 pos_x = Map X of Player
◆Control Variables : #0002 pos_y = Map Y of Player
◆Control Variables : #0004 enemy_x = Map X of This Event
◆Control Variables : #0005 enemy_y = Map Y of This Event
◆If : Player is facing Down
  ◆If : pos_x = enemy_x
    ◆If : pos_y < enemy_y
      ◆Set Movement Route : This Event (Wait)
       :                  : ◇Wait : 180 frames
      ◆
    : End
    ◆
  : End
  ◆
: End
```

Figure 11. Code for the player's light detection

C. Extending Puzzle Archetypes

After the verification and validation of the refactored codes, an extension was performed for the respective Password, Sliding Block and Six Lights puzzles by the usage of available RPG Maker plugins. In general, it is a common action into the development community, where personalized libraries that allow the creation of advanced mechanics in several RPGs are available to functionally extend them. As a result, new aesthetic and gameplay perspective about developed puzzles can be provided, extending their functionalities with new game mechanics and dynamics possibilities.

As an example, together with the logic applied in the previous puzzles, a plugin that allows the development of a mechanics associated to the light detection was used (Figure 10). The featured mechanics consists of making constant assessments of monster positions in relation to the player, that has a light source attached to it (flashlight) that highlights the area that the player is facing. Once the monsters are in reach of the light cast by the player, they briefly stop following it (Figure 11).

III. Conclusion and Future Work

This paper presented a study of puzzle archetypes for RPG Maker MV in an analysis, evaluation, refactoring and extension perspective. It is an important research that shows different types of RPG puzzles strategies, as well as development difficulties and available opportunities for the application of reusable approaches in the production of RPG puzzles.

In fact, regarding the identification and analysis of developed puzzles by the RPG Maker MV community, puzzles that do not use native framework resources, in this case derived from available plugins, still need to be studied and properly modeled. They represent a hack to avoid programming limitations of the native framework, providing as a result a great variety of archetypes that were not be identified and analyzed yet.

Regarding the refactoring perspective, it was noticed that there is no common effort for the application of heuristics and memory optimization in the community codes. The study of these codes also reveals the prioritization of aesthetics and functionality over efficiency and high performance, which can lead to changes in the player experience in real time execution. Therefore, optimization and reuse opportunities are opened for RPG puzzles in the RPG Maker environment.

Finally, considering the extension perspective, by combining with available RPG plugins, many types of game mechanics and dynamics can be provided to the identified puzzle archetypes, introducing a new aesthetic and gameplay perspective about developed puzzles. In this sense, there is an interesting research opportunity to explore and expand the cataloged dynamics regarding puzzles and existing RPGs, and to define configuration features for the RPG puzzle domain to allow the production of RPG puzzles in large scale.

References

[1] M. J. Wolf, *The medium of the video game*. University of Texas Press, 2001.

[2] K. G. P. A; and S. K, *A survey of np-complete puzzles, international computer games association journal*, 2008.

[3] J. G. Cover, *The Creation of Narrative in Tabletop Role-Playing Games*. McFarland Company, 2010.

[4] M. Sayer. (2017). The surprising explosion of rpg maker on steam, [Online]. Available: https://www.pcgamer.com/the-surprising-explosion-of-rpg-maker-on-steam/. (accessed: 19/07/2018).

[5] D. Perez, *Beginning RPG Maker MV*. Apress, 2016.

[6] M. Conforti. (2013). Categories of puzzles, [Online]. Available: http://blog.rpgmakerweb.com/tutorials/categories-of-puzzles/. (accessed: 11/07/2018).