T2GAME: A New Domain Specific Language to Model Behavior Patterns on Games

Ismael Posada Trobo *[a,b], Vicente García-Díaz [a], Jordán Pascual Espada [a], Rubén González Crespo [c], Enrique Herrera-Viedma [d,e]

[a] *MDE Research Group, Department of Computer Science, University of Oviedo, c/Calvo Sotelo s/n, 33007, Oviedo, Asturias, Spain*

[b] *CERN IT/OIS, 1211 Geneva 23, Switzerland*

[c] *School of Engineering, Universidad Internacional de La Rioja - UNIR, Avda. Rey Juan Carlos I, 42, Logroño, La Rioja, Spain*

[d] *Dept. of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain*

[e] *Department of Electrical and Computer Engineering, Faculty of Engineering, King Abdulaziz University, Jeddah, 21589, Saudi Arabia*

**Corresponding author: Tel.: +34 616462523*

E-mail addresses: ismael.trobo@cern.ch

C/ Calvo Sotelo s/n Facultad de Ciencias 3º OOTLab , 33007, Oviedo, Asturias, Spain

Abstract

Game generation applications from Domain Specific Languages (DSLs) have prevailed during the last few years. For example, in the Gade4all Project, a game code generator application, was developed in order to allow users to create different computer games, using domain specific languages, allowing them to define their ideas with a programming misknowledge. Gade4all allowed us to define the whole process of a computer game, without going into complex parts of the game, as for instance, behaviors of Non-Player Characters (NPC). Based on this, in this work we go a step further and we provide users the necessary tools so that, with a non-extensive or even low notions of programming knowledge, users are able to provide a customized behavior for the NPCs through an application generator. T2Game is a proposal regarding how to make and deploy complex behavior patterns, proposing a design which allows users with a lack of programming knowledge to define complex behavior and interaction patterns of enemies in a simple way.

Keywords: DSL, MDE, Computer Games, Behavior Patterns, Modeling Tools, Code Generation.

I. INTRODUCTION

From the beginning of computer games development, provided generation code tools contained a strong programming factor, which limited to some extent the access or the elaboration process to a part of the society of this kind of software. By this way, only a selected and qualified group of people were able to develop computer games, as they had the required knowledge about it, especially regarding code programming, increasing exponentially the complexity of these kind of systems.

Gade4all [1], a computer games development environment, was developed in order to allow users to create simple 2D computer games from a Domain Specific Language (DSL). The main goal of that project was to provide the necessary tools for computer games creation in a simple way to users with a lack of programming knowledge, as compared with the way third-party tools provided at that time. It greatly facilitates the general workflow of the solution, but with some limitations.

Computer games show up many customized aspects, being part of the whole process of computer games creation. Creating or modifying these aspects through commercial editors could be relatively easy, depending on the knowledge of the user in terms of programming, although normally these editors facilitate the labors of creating a game by raising the level of abstraction. However, there are other aspects we could consider complex for which editors does not provide a clear help, such as configuring behaviors of the NPCs. Over the last few years, game development has been largely influenced by Artificial Intelligence (AI) [2] [3] [4]. AI plays an outstanding role in modern video games by making them feel both more realistic and funnier to play, working alongside the game logic, invisible to the players who enjoy the resulting character behaviors [5]. It has been developing and focusing on different game categories, such as platform games, First-Person-Shooter (FPS), Role Playing Games (RPGs), etc. [6], being one of the biggest challenges to create NPCs behavior [7]. It can delimitate the whole gameplay in terms of quality and interaction. Creating an understandable abstraction of these aspects is not easy for common users of such tools, as we could leave aside some important parts that for us are not important, but they really are. All this suggests that the configuration of enemies' behaviors can be much more optimized if we create the proper abstraction.

Thus, T2Game, our proposal, is born to help users to design and create behavior and interaction patterns based on a DSL, serving as a base for the reuse of code in an automatic way, improving the efficiency and solving typical problems of software development by the use of Model Driven Engineering (MDE) paradigm [8]. MDE increases the level of abstraction through a design, based on formal models, understandable by users with a lack of programming knowledge, although with a knowledge of the specific domain in which the DSL focuses.

In this paper, we discuss the method we adopted to solve the lack of proposals regarding the modeling of behavior patterns on games. The rest of this paper structured as follows. Section II overviews the background on this topic, introducing the user to several ways of configure behaviors and their usage difficulties. Section III introduces the user to our proposal, T2Game, and the way it works. Section IV shows how T2Game can be used to address the modeling actions with real life samples. Section V evaluates the proposal and finally, Section VI presents the conclusions and the future work.

II. BACKGROUND

In 90s, computer games development experimented a boom, being the proliferation of the games creation tools massive. Practically the majority of these tools began to follow a philosophy of fast development without delving into any special feature, but a vast knowledge of the domain was required. These developments were used by a unique person, an expert in many fields concerned to computer games.

With the raise of industry in the XXI century, these developments begin to be used by inexperienced users. No much programming knowledge about development was needed and users started to do their own developments. Games industry realized the importance of development for these users and evolved offering two alternatives: computer games editors based on wizards, for modeling elements of a computer game and specific frameworks for, basically, experienced users. As opposed, the development time consumed by these applications had no experimented big changes since then, being extremely high [9].

Some of these used tools are, inter alia: Game Maker Studio [10], Construct-2 [11] and Stencyl [12].

Game Maker Studio is a games development tool based on an interpreted programming language called GML. It has been designed to allow users to easy develop computer games for different platforms such as iOS, Android or PC. This tool uses the philosophy of drag and drop for configuring interactions and the relationships between the objects included in the game. The main goal of this tool is to minimize the creation of the game elements avoiding code writing.

Relative to enemies behavior, this tool brings up plenty possibilities of creation, based on the concept of associated actions to events.

Contruct-2 is a 2D games editor developed by Scirra, especially for web platforms (HTML5), intended by users with a lack of programming knowledge. It uses drag & drop philosophy using a behaviors editor with the particularity of adding functionality even with a logic system based on.

Behavior system is based on predefined packages in order to add more functionality, with a bunch of customized properties. Essentially, it could be considered as 'shortcuts' in order to improve productivity, also known as 'time-savers'.

Stencyl is a games development tool created by Stencyl Works which provide designers a graphic editor in order to develop games for different platforms. Apart from the possibilities that brings up about configuring actors, scenes, etc., it has a particular way to define actors behaviors, based on linked pieces of a puzzle. These 'pieces' are called Behaviors, reusable entities that are joint to actors or scenes. At a glance, a programmer could believe that it sets a kind of pseudo-code.

Delving into configuring behaviors could require certain programming and algorithm knowledge.

All of these tools offer a similar range of functionalities and operations, having its own particularities, defined by different rules and parameters, especially in terms of behaviors configuration, but similarities means same limitations. Any user could define different elements within game, but once we want to go a step further configuring complex behaviors, for instance, mixing them, these tools begin to limit to some extent the behavior creation, being necessary a depth knowledge of what we are doing.

Those limitations are directly related with the proposal we introduce in the next section, as we try to define a known and interpreted common way of building enemies behaviors by users.

More alternatives for computer games development are the use of frameworks, such a **Unity 3D** [13] or **Blueprints** [14] from Unreal. These are specifically designed to facilitate the managed of a particular problem through coding, as it is on Unity 3D, or through diagrams and coding, as it is on Blueprints. Both show different ways to achieve same solutions, but it is more intended by experienced users.

There are many researchers focused on the design of tools or platforms for modeling through MDE, as shown in Krogmann and Becker [15], comparing code development to software developed using the MDE approach with same functionality.

Gade4all project (Fig.1) is created with MDE approach in mind, proposing the use of models for creating games from the beginning, allowing users with non-technical knowledge only focusing over the creative area.

Fig. 1 Gade4all editor

Other proposal defends the use of working together both designers and developers on a more productivity way using MDE, as it is said in Furtado and Santos [16].

Jaime et al. [17] try to reduce the cost of traditional games development with a new approach of modeling the game loop of a game called VGPM.

Carton [18], with the use of Java Emitter Templates (JET) technology for coding generation, presents an approach in order to manage the complexity based on a combination of aspect-oriented development and model-driven engineering techniques.

Some works related to behaviors applied to games can be seen in Cardamone et al. [19]

III. T2Game Proposal: A New Domain Specific Language to Model Behavior Patterns on Games

The main focus of this work is the creation of a graphic domain-specific modeling language to allow users to model the behavior of enemies in game. In order to meet this objective and check the suitability of our solution, we have developed a prototype which is structured as follows:

- Metamodeling
- Model & Graphic Syntax: editor

- Import to JET code generator: JET Templates
- *android_platform_miw* Android Project

A. Metamodeling

The metamodeling defines the elements of the modeling language (meta-classes), their relationship (meta-relations) and their restrictions. That is, it defines the domain which we are going to act with. This domain is the enemies' behavior patterns modeled in platform games.

Metamodeling is the key of the MDE, and it is much needed for creating domain specific languages, model validations, model transformations, artefact generation and tools integration [20]. It defines the abstract syntax and the static semantic, (e.g., by using the Eclipse Modeling Framework (EMF) tools through *Ecore).*

In order to create the domain language, this metamodeling will be composed by a set of entities, described below. Those entities have common names extracted from a natural language, in order to provide better understanding by users. The list of entities are described below:

- *Behavior*
- *When*
- *Condition*
- *Action*
- *Otherwise*

Behavior: it is the main node, grouping the rest of entities of the metamodeling. It is composed by zero or many '*When*' entities (idem for '*Action*' entity).

When: entity whose main goal is to set a condition in the model processing. It carries a group of predefined identifiers, allowing users to choose between them. It is composed by '*Condition*', '*Otherwise*' and '*Action*' entities. This entity could appear alone, without dependence on others.

Condition: this entity is directly associated with the *When* entity, which set what action is going to be performed through an identifier called '*idCondition*'. If a *When* entity is set, *Condition* entity must appear.

The valid associated identifiers for the *Condition* entity are the following in this version of the language:

- *IsThereWall:* this identifier allows enemies to check whether an obstacle prevents or not to enemies to carry on with their movement.
- *playerIsNear:* this identifier allows enemies to know if the main character is close. The default value is 150 px.
- *playerHasEnoughLife:* this identifier asks for the health of the main character, in order to do some actions related to this health.
- *playerHitsMe:* this identifier checks if the main character has collided with an enemy. This is obtained through intersections of rectangles.

Otherwise: optional entity associated to *When* entity. If *When* entity does not satisfy the condition, *Otherwise* entity gets into play. It could appear (n-1 < x < n) times, being '*x*' the *Otherwise* entity and 'n' the *When* entity.

Action: this entity can act independently, setting an action within behavior. These actions are predefined by a finite set of themselves through the '*idAction*' variable (or action's identifier).

Valid '*idAction*' identifiers are the following:

- *stop*: this identifier sets the movement of the enemy to zero, avoiding its movement.
- *turnAround:* this identifier sets the inverse of the current movement. If the enemy is moving from right to left, this will set the movement from left to right, and vice versa.
- *moveHorizontal:* this identifier sets the movement to true. That is, start the process of move an enemy. It is the opposite of the *stop* identifier.
- *startFire:* this identifier allows enemies to start shooting to the main character. By default, it has its own fire span in order to avoid multi shots.
- *stopFire:* this identifier is the opposite of the *startFire* identifier. If enemy has activated its *startFire* flag, this will cancel it.
- *killPlayer:* this identifier allows enemies to kill the main character, or in other words, will terminate with the

main character health. The game is over when this identifier happens.

If the *Action* entity is associated to *When* entity, this *Action* entity will be processed when as long as the *Condition* entity of the *When* entity is satisfied. However, if it is associated to the *Otherwise* entity, it will be processed when *Condition* entity from *When* entity is not satisfied.

These pre-associated identifiers refer to the associated code used in the enemies' behavior part at the deployed application. Their election and identification is based on actions which commonly appear in platform games. Every single identifier constitute the method calling, after applying a set of modifications and refactoring previously. These associated codes could be easily added by programming experts, in order to provide more dynamism and increase the number of actions and conditions provided to generate more behavior patterns, and allow more characteristics.

B. Model & Graphic Syntax: editor

Once we have defined the language, it is time to design the model. This model is an abstraction focused on solving problems of a specific applications. In our case, this is the platform games behavior patterns. This model uses defined elements from metamodeling in order to specify the behavior pattern of an enemy within platform games. These models are used for increasing productivity and compatibility between different systems, simplifying the design phase, exalting work team communications, as it said in Schmidt [21].

To do that, we are providing two different editors, looking for being intuitive for users at the time of creating model. One is based on '*tree node*' philosophy, contributing with parent nodes, child nodes, etc., and the other one is based on the '*drag & drop*' philosophy, made by the tools Sirius system provides.

For the first one, we have to place over the parent node, 'Behavior', right-clicking and adding the proper children and the proper properties to each entity. It can be seen in Fig. 2.

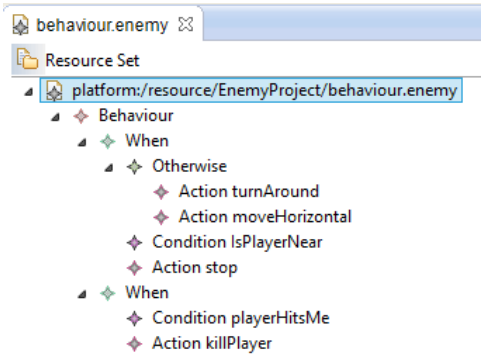

Fig. 2 Sample of Tree Node editor

For the second one, this editor is developed with Sirius, and it follows the '*drag & drop*' philosophy. It is composed by a palette with a range of elements, placed on the right, as it can be seen in Fig. 3. These elements represent the entities previously defined. In addition, it allows to specify what actions are going to be joined with both different entities (*When* and *Otherwise*), similar to the first editor, but in a more 'colorful' and graphical way.

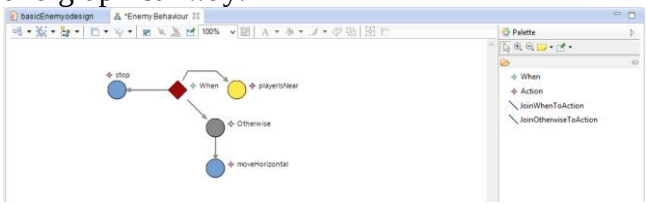

Fig. 3 'Drag & drop' editor, developed with Sirius

Both editors will generate a XMI version 2.0 specification, as shows Fig. 4, used by default by EMF technology for data persistence. This specification will be responsible for communicating both the model design and the JET code generator.

```
<Enemy:Behaviour xmlns:xmi="http://www.omg.org/XMI" xmlns:Enemy="Enemy" xmi:version="2.0">
  <whens>
    <otherwise>
      <actions idAction="turnAround"/>
      <actions idAction="moveHorizontal"/>
    </otherwise>
    <condition idCondition="IsPlayerNear"/>
    <actions idAction="stop"/>
  </whens>
  <whens>
    <condition idCondition="playerHitsMe"/>
    <actions idAction="killPlayer"/>
  </whens>
</Enemy:Behaviour>
```

Fig. 4 Sample of XMI specification

C. Import to JET code generator: JET Templates

JET code generation is a Model-to-Text (M2T) engine based on an EMF model. In JET, a variety of templates are defined, called JET templates. These templates allow JET code generator to generate automatic code, as Java, XML, etc.

Code generation is a fundamental part in MDE, since these templates define the implementations

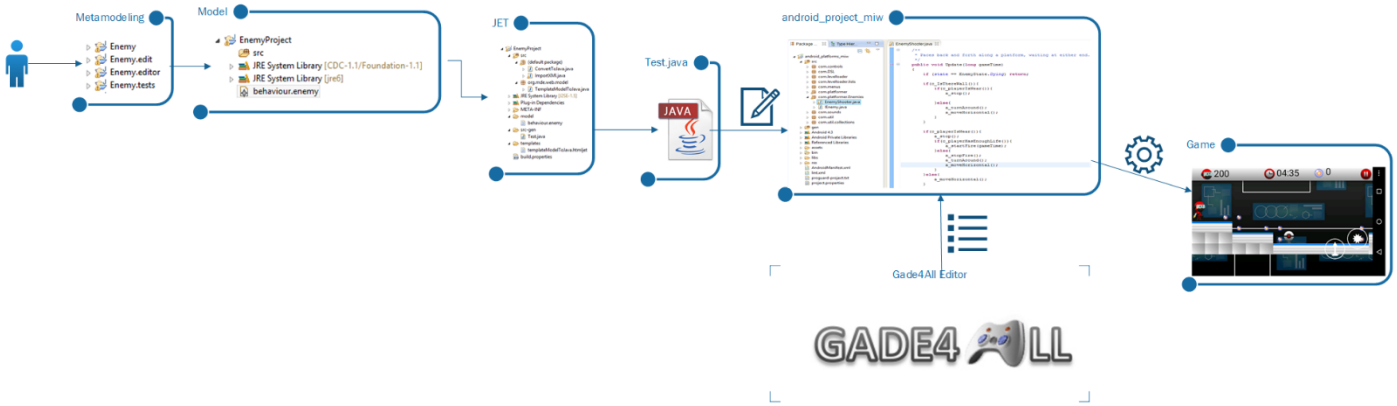

Fig. 5 Overview of our solution

created by after the transformation step. In our solution, this JET engine receives a XMI file, and transforms it into Java code. This Java code is responsible for setting the proper actions in the enemy behavior, being ready for the last execution: our target application.

D. android_platform_miw Android Project

Our target application is a 2D platform game entirely developed for Android in Java code. The creation of this application has been carried out by the project Gade4all. The whole process can be seen in Fig. 5.

IV. USE CASE: MODELING A REPRESENTATIVE BEHAVIOR PATTERN

Once the proposal has been introduced, the next step is to show how the whole videogame definition is done using it. In this case, the presented tools will be used to define and generate an example of the '*Goomba's*' behavior, present at Mario Bros. game developed by Nintendo®. For this first case we are going to use the '*Tree node*' editor.

Fig. 6 shows the flow chart we are following in order to set the behavior to our enemies.

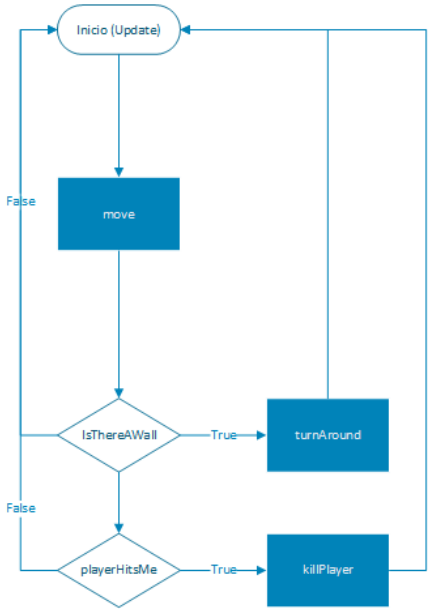

Fig. 6 Flow chart of use case

In order to achieve this, we will generate a platform game using the Gade4all editor. Now, it is time to set the behavior. Users must place over the *Behavior* node on our editor, and start to adding nodes through right-clicking the contextual menu, creating new children, achieving the model as it is shown on Fig. 7.

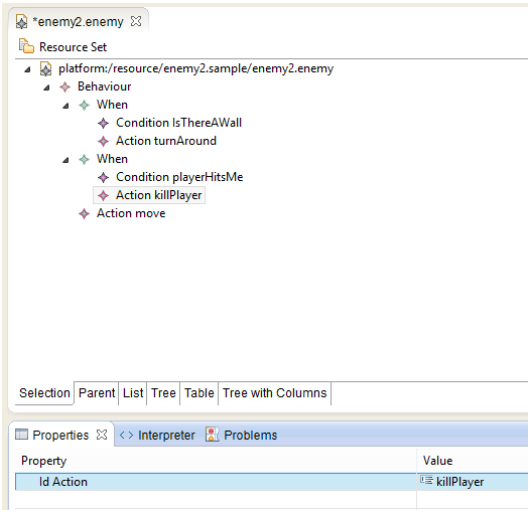

Fig. 7 Use case set with Tree node editor

As Fig. 8 shows, we could take a look to the XMI specification, placing us over the left menu and open the model up with the Text Editor feature. This XMI, as we have previously defined, will be used by the code generator.

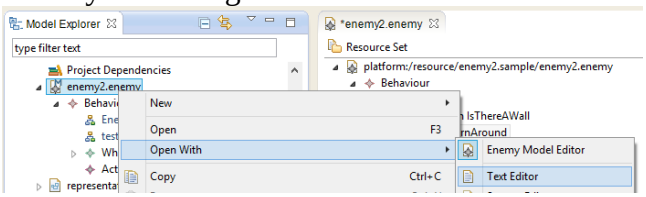

Fig. 8 Overview of XMI Specification

With the generated specification done, code generator must be executed. Through a JET template, this code generator will transform that specification into Java code, generating a Java file. This Java file contains the update method used to update the behavior of the enemy within game. In order to apply this code to the target application, this operation must be performed by hand, but obviously in a release solution, should be done automatically. In the target application, EnemyShooter.java class will be affected.

Finally, in order to see the result, the user must import the generated projects into the corresponding integrated development environment, also known as IDE. In this case, the application project must be imported into Eclipse with the Android Development Kit plugin installed.

When the projects have been imported there are two possible ways to execute them and check their behavior: using the emulators included with the development environment or attaching a physical device to the computer and deploying the videogames on it.

Another way to achieve this could do through Sirius editor, also developed by us. With the same flow chart as it is shown in Fig. 6, the process of adding elements is quite similar. In Sirius editor, users have to accede to the elements provided by the palette, placing on the right side as shows Fig. 9, and then drag & drop the elements over the viewer.

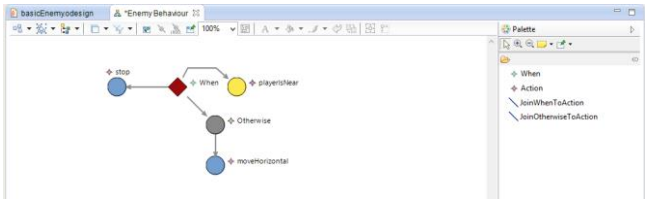

Fig. 9 Overview of Sirius editor

Once we have dragged and dropped the elements, it is time to set the suitable properties, based on the properties previously annotated on Section III. In addition, it is necessary to join the elements through the lines, specifying the order they are going to execute at. An overview of the process made with Sirius editor is shown in Fig.10.

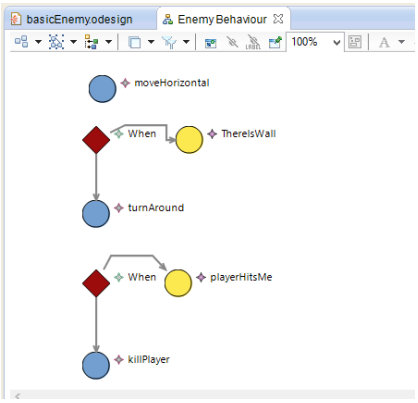

Fig. 10 Use case with Sirius editor

Once it is defined, the XMI specification will be generated. The following steps, in order to deploy the solution, are similar as defined in the previous editor.

V. Evaluation

Once we have set what is the main goal we are following in our proposal, and having explained the whole functioning, it is time to evaluate carefully respect to third-party tools present in the market about definition of behavior patterns. At the time of evaluation, it is not clear what is the best method to affront this, as there are many

factors that carry weight into edition of this kind of behavior patterns.

In order to do this, we propose an evaluation system based on those aspects we believe they affect into the creation complexity of doing the proper operations to get a behavior. We have 5 elements clearly differenced that they take part over the whole process of making patterns. They are:

- Written characters.
- Used/Created classes.
- Provided method(s) by framework or editors used by helping in the building phase.
- Used nodes or boxes.
- Affected properties they have been created for some reason.

Having defined the elements, we have set a kind of weights that they delimit the complexity of the operation. For instance, we consider that use a node has less 'weight' than use a provided method by frameworks, as to reach both elements, there are clearly differences in term of complexity.

Obtained results for the previous use case can be seen in Table 1. Weights of different operations are shown between brackets:

	Characters (0,3)	Classes (0,2)	Framework Methods (0,3)	Nodes (0,4)	Properties (0,6)	Total
Unity	69,9	0,2	0,9	0	1,2	72,2
Game Maker	21,3	0,2	0,3	0	1,2	23
Stencyl	4,5	0	0	1,6	3	9,1
T2Game	6,6	0	0	0,8	1,2	8,6
Blueprints	10,8	0	0	2,8	3,6	17,2

Table 1 Use case results

Fig. 11 shows a chart with the obtained results in a logarithmic scale due to data covers a wide range of values:

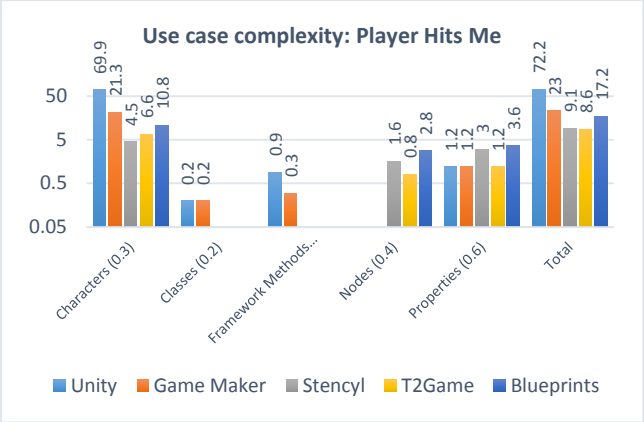

Fig. 11 Use case complexity

As we can see, our proposal presents a slight improvement in terms of creation complexity compared to solutions like *Stencyl*. This improvement is pretty much 0.42% on average. This tiny difference is concerned to *Stencyl* is a tool fairly similar to ours, in principle because of *Stencyl* presents a way of doing these kind of behaviors similar to ours. The main part concerned to this difference is the puzzle part *Stencyl* presents, as well as the use of nodes of our solution compare to *Stencyl* solution. As it can be seen in Fig. 12, the part concerns to edit the behaviors, is done by a puzzle-like process. In this case, at the behavior configuration, we have to specify what actors are getting involved in the behavior pattern, that is, we have to specify most of the properties we want to use into behavior and its relationship. Here the first difference appears, as we provide a language that internally has their own identifiers so that the proper actors 'know' what behavior has to perform. We know beforehand how actors are affected since in most cases these properties are predefined and assigned at the beginning of creation.

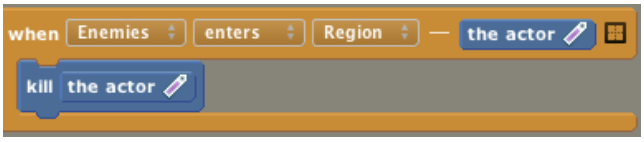

Fig. 12 Use case made by Stencyl tool

On the other hand, extending behavior patterns in *Stencyl* tool could not be easy, as users would feel data overload as it does not follow an understandable and continuous flow. Comparing with our solution as we can see in Fig. 13, we

always show the flow that a behavior pattern is following. In *Stencyl*, the way to do as a puzzle could be useful for small behavior patterns, but once these behaviors are increasing the elements that get into play, the level of difficulty is increased at the time of configuring different parts. The number of properties used tends to grow, while for our solution all these properties represents a finite set, being uniquely increased up depending on the language specification. This obviously could be proved by doing usability tests in the future, comparing both *Stencyl* tool and our solution and see how users act in both situations.

Thereby, we have clear evidences that our proposal have improved some parts of the creation we consider they are difficult for understanding by users.

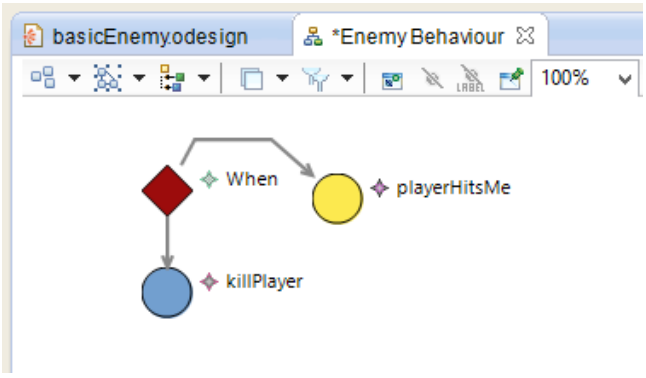

Fig. 13 Use case made by T2Game editor

Respect to other solutions as *Blueprints*, this tool does a major use of nodes at the time of setting the different events they are taking part into game. Thus, having a look to the obtained results, our solution is 11% simplest than *Blueprint* on average in terms of creation complexity. Basically this difference is concerned as *Blueprint* needs to set all the variables in nodes in most cases, which is not necessary on ours.

Having a look to the results, if we compare results in different editors or frameworks as *Blueprints*, *Game Maker* or *Unity 3D*, we notice that differences increase when we are setting a major number of options, overall in framework systems. This could seems obvious, but it is a way to say that editors way of creation could be relatively simplest than programmatic way, as measurements are shown in Table 1. Summarizing, programmatic way of doing these kind of behaviors would increase faster the creation complexity than the editors way permit.

VI. Conclusions and Future Work

In this paper we present a novel proposal that allows users to model the behavior patterns logic of enemies within the game: a new specific language to model behavior patterns. The designed language was created based on our methodology which is being improved and refined by a research group with new ways of achieving different use cases. Using this methodology we will get further information from domain experts in order to detect more elements and fundaments of the enemies' behavior logic, and their relationship. These specific elements will be handled by a couple of editors, with the proper restrictions and the specific properties. This will be the way we will achieve that users with a lack of programming knowledge could model a process that defines the whole logic of an enemy behavior.

The behavior pattern process modeling through DSLs is based on the combination of both elements and properties set in the editor for our first approach. This language is not as powerful as other programming languages used in games development due to some limitations and restrictions provided in order to facilitate its managing by users. In compensation, the expressiveness of the defined language is fairly significant, accomplishing the full coverage of the different and typical phases an enemy behavior presents within a game. This allows users define enemies' behavior patterns in an easy manner, as we have demonstrated previously, always keep in mind that we are in front of inexperienced users.

Thus, our approach is capable of reducing the creation complexity and the time spent experimented by users over the whole phase of behavior creation of enemies, improving the efficiency of common game development tools, as well as facilitating the way of creation.

The future work regarding this investigation is oriented to improve and optimize our proposal. It is divided in the following areas:

- New improvements and optimizations about the presented structure with the aim of setting them more affordable, simpler to modeling and with the enough abstraction level in order to allow variations of it.

- Diversify behavior patterns making them more specific, so that each enemy has its own. Furthermore, it will be interesting to parametrize the behaviors, as well as include fuzzy logic in this process.
- Promote the use of behavior patterns not only for enemies, but for the different actors present in game, such as items, main characters, final enemies, etc.
- New dive into behavior patterns of 3D games. A pattern must be applied to either 2D game or 3D games.
- New research about new behavior patterns feedback, acting as conditions allow, learning from environment. At the same time, it will be interesting that behaviors act as emotions allow.

ACKNOWLEDGEMENTS

This paper has been partially developed with the financing of FEDER funds in TIN2013-40658-P and Andalusian Excellence Project TIC-5991.

BIBLIOGRAPHY

[1] E. Rolando Nuñez-Valdez, O. Sanjuan, B. C. Pelayo García-Bustelo, J. M. Cueva-Lovelle y G. Infante Hernández, Gade4all: Developing Multi-platform Videogames based on Domain Specific Languages and Model Driven Engineering, Oviedo: International Journal of Artificial Intelligence and Interactive Multimedia, 2013.

[2] X. Tu y D. Terzopoulos, Artificial fishes: Physics, locomotion, perception, behavior, Orlando: In Proceedings of SIGGRAPH, 1994.

[3] J. Funge, X. Tu y D. Terzopoulos, Cognitive Modeling: Knowledge, Reasoning and Planning for Intelligent Pedestrians, Los Angeles, 1999.

[4] S. Russell y P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education, 1995.

[5] M. Pirovano y P. L. Lanzi, «Fuzzy Tactics: A scripting game that leverages fuzzy logic as an engaging game mechanic.,» *Expert Systems with Applications,* vol. 41, nº 13, pp. 6029-6038, 2014.

[6] T. S. H. G. Vidaver, Flexible an Purposeful NPC Behaviors using Real-Time Genetic Control, Vancouver, 2006.

[7] C.-N. ZHOU, X.-L. YU, J.-Y. SUN y X.-L. YAN, Affective Computation Based NPC Behaviors Modeling, Proceedings of the 2006 IEEE/WIC/ACM Internation Conference on Web Intelligence and Intelligence Agent Technology, 2006.

[8] S. Kent, «Model Driven Engineering,» *IFM '02 Proceedings of the Third International Conference on Integrated* , nº ISBN:3-540-43703-7, pp. 286-298, 2002.

[9] J. Solís-Martínez, J. Pascual Espada, B. C. Pelayo G-Bustelo y J. M. Cueva Lovelle, BPMN MUSIM: Approach to improve the domain expert's efficiency in business process modeling for the generation of specific software applications, Oviedo: Elsevier, 2014.

[10] «Game Maker Studio,» YoYo Games, 15 Noviembre 1999. [En línea]. Available: https://www.yoyogames.com/studio. [Último acceso: Mayo 2015].

[11] «Construct 2,» Scirra LTD, 4 Febrero 2011. [En línea]. Available: https://www.scirra.com/construct2. [Último acceso: Mayo 2015].

[12] J. Chung, «stencyl,» Stencyl, LLC, 31 Mayo 2011. [En línea]. Available: http://www.stencyl.com/. [Último acceso: Mayo 2015].

[13] D. Helgason, «unity,» Unity Technologies, 30 Mayo 2005. [En línea]. Available: https://unity3d.com/es. [Último acceso: Mayo 2015].

[14] «Blueprints Visual Scripting,» Unreal Engine, 1998. [En línea]. Available: https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html. [Último acceso: Mayo 2015].

[15] K. Krogmann y S. Becker, A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor, Karlsruhe, 2007.

[16] A. W. B. Furtado y A. L. M. Santos, Using Domain-Specific Modeling towards Computer Games Development Industrialization, Pernambuco, 2006.

[17] Martinez, Solis Jaime et al., «VGPM: Using business process modeling for videogame modeling and code generation in multiple platforms.,» *Computer Standards & Interfaces,* vol. 42, pp. 42-45, 2015.

[18] Carton, Andrew; et al., «Aspect-oriented model-driven development for mobile context-aware computing,» de *Proceedings of the 1st International Workshop on Software Engineering for Pervasive Computing Applicaitons, Systems and Environments*, Dublin, IEEE Computer Society, 2007, p. 5.

[19] Luigi Cardamone et al., «Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture,» *Expert Systems with Applications,* vol. 40, nº 16, pp. 6447-6458, 2013.

[20] J. Pascual Espada y V. García Díaz, Ingeniería dirigida por modelos, Oviedo: UNIOVI, 2014.

[21] D. C. Schmidt, «Model-Driven Engineering,» *IEEE Computer*, vol. 39, nº 2, pp. 25-31, 2006.