Phylogenix: Bringing phylogenetics to Unity

Samuel Navarro, Antonio Iglesias, Francisca Pérez, Carlos Cetina, Jaime Font
{snavarrod,aiglesias,mfperez,ccetina,jfont}@usj.es
Universidad San Jorge. Escuela de Arquitectura y Tecnología
Zaragoza, Spain

ABSTRACT

Game Software Engineering addresses the software part of creating video games, with game engines like Unity serving as foundational tools for development. A major challenge in game development is creating new content, resulting in the exploration of Procedural Content Generation techniques. However, these techniques often lack formalized variability, hindering the transition to a Software Product Line paradigm. This paper presents Phylogenix, a Unity plugin that leverages phylogenetic analysis for a family of video game content where variability is not formalized. The resulting phylogenetic tree has potential to understand the variability and even release latent family members. A video of the tool is available at https://youtu.be/TxaQxGXrtqI

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; **Software reverse engineering**.

KEYWORDS

Phylogenetics, Software Product Families, Game Software Engineering, Procedural Content Generation

ACM Reference Format:
Samuel Navarro, Antonio Iglesias, Francisca Pérez, Carlos Cetina, Jaime Font. 2024. Phylogenix: Bringing phylogenetics to Unity. In *Proceedings of 28th ACM International Systems and Software Product Lines Conference (SPLC'24).* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The video games industry has become the largest entertainment sector, surpassing in revenue music and cinema together [23]. Moreover, half of the software developers in the world are involved in video game development [31]. Video games are complex pieces of software that combine artistic and technical components to deliver a "fun" experience for players. Game Software Engineering (GSE) has emerged to address those particularities [1, 11].

Game engines are the main tool used by the industry to create video games. A game engine is a development environment that includes the foundations for any game (e.g. graphics and physics of the game) as reusable components. They also include a set of tools that help the developer to accelerate the development, and reduce duplication of code and effort [23]. Unity engine[1] is one of the most used game engines in the industry.

One of the bottlenecks in video game development is the creation of new content and many works from the GSE community propose Procedural Content Generation (PCG) techniques [15, 28]. However, those techniques produce variants of existing products, without formalizing variability among the family, and thus they do not promote the shift to a Software Product Line (SPL) paradigm.

Recently, Chueca et al. [9] applied the concept of phylogenetics from biology to the industry of video games to generate new content from an existing family. They introduced that phylogenetics can also be used as a form of Domain Analysis [22], which can serve as a first step towards formalization of the variability. Phylogenetic analysis results in a tree that offers an ordered and structured classification of the family of products analyzed.

In this work, we present Phylogenix, a tool developed for the Unity engine that enables game developers to analyze existing content using phylogenetics. To this end, Phylogenix analyzes and compares, pair by pair, all the elements of the given game, to generate a tree representation of the content family.

The rest of the paper is structured as follows. Section 2 presents the basics of phylogenetics in biology. Section 3 presents game engines and particularities of Unity engine. Section 4 presents how Phylogenix applies phylogenetics to any video game developed in Unity. Section 5 presents four case studies in which we have validated our tool. Section 6 presents the related work. Section 7 concludes the paper.

2 PHYLOGENETICS

In biology, authors refer to phylogenetics as the study of phylogenetic relationships, and their representations (phylogenetic trees), to clarify evolutionary phenomena [17]. The relationship studies concern different species or groups of individuals of the same species, which are referred to as *taxa*. Each *taxon* has a genome that describes the state of each of the characters of the individual, meaning each of the traits or features that distinguish one *taxon* from another [3]. To elucidate these relationships, it is necessary to measure the difference among the *taxa*: their genetic distance. To this end, each pair of *taxa* is compared using a Genetic Distance Matrix, a square matrix that represents the genetic distance among *taxa* [30]. Finally, inference techniques are applied, using the genetic distance matrix as input, to produce a phylogenetic tree [3]. This phylogenetic tree allows researchers to study the relationships in the set of *taxa*, according to genetics.

Charles Darwin claimed that life is the product of descent from common ancestors and, to communicate that idea, Darwin introduced the metaphor of the *tree of life* [12]. From that analogy, Baum

[1] https://unity.com/

SPLC'24, September 2-6, 2024, Luxembourg

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

et al. [4], initiated the Tree-Thinking movement, with the affirmation that anyone, even without knowledge about phylogenetics or biology, could interpret trees and use them for organizing and classifying knowledge. From that, the most common representation for the results of a phylogenetic inference process is a phylogenetic tree, also known as *phylogram*. It is an undirected graph representation with nodes (representing *taxa*) and links (representing the ancestry among *taxa*). It does not only represent the relationships but also the difference among the *taxa* using the edges' length [21].

There are multiple representations in which a *phylogram* can be presented. The most common types are rectangular and circular. The bottom part of Figure 1 shows an example of each type.

3 UNITY ENGINE

Game development comprises a set of complex processes that require the effort of heterogeneous teams. These processes include tasks like creating the story, developing behaviors, or designing the characters and scenarios where the game will take place.

Every element in the game is considered a *GameObject* (GO). For example, in a shooter game (i.e. a video game where the player must shoot at multiple enemies), the development team will need to create different types of GOs to represent weapons, enemies, ammo, and war scenarios. GOs are the main building block in Unity, and enable developers to populate the game easily including both logical (e.g. how the bullet behaves moving across the scenario) and visual elements (e.g. the appearance of the bullet). Moreover, GOs are composed of different types of components, such as Transforms, Colliders, Meshes, or MonoBehaviours. For example, a GO can have multiple MonoBehaviours (each MonoBehaviour associates a script to the GO), and those MonoBehaviours can coincide between different GOs, or can be specific of a single GO. In addition, multiple copies of each GO could be needed (e.g. many bullets when shooting, or many enemies to shoot at).

To reduce the complexity of the creation of GOs, game engines provide mechanisms like Unity *Prefabs*, that enable the creation of collections of reusable, configurable, and modular GOs [27]. *Prefabs* are templates for GOs; making an analogy, a *Prefab* could be considered like a *class*. Using a *class* we can create multiple *instances*, and using a *Prefab* we can create multiple GO instances. Continuing with the example of the shooter game, we will have a *Prefab* of a bullet that will be instantiated every time the player pulls the trigger of their weapon.

4 PHYLOGENETICS IN UNITY

Phylogenix enables developers to analyze *Prefabs* and generate a phylogenetic tree of any video game developed in Unity. Figure 1 shows the flow of such phylogenetic analysis of *Prefabs*.

First, as shown in Figure 1, *Prefabs* of the game are analyzed, searching for the components used in the project. In the *Prefabs* of the figure we have Transform, Collider, Mesh, MonoBehaviour 1, MonoBehaviour 2, and MonoBehaviour 3. After that analysis, a sequence of 1s and 0s is generated for each of the *Prefabs*, indicating the presence or absence of components. That sequence represents the genome of the *Prefab*.

Second, the genetic distance between each pair of *Prefabs* is calculated to generate a distance matrix. Phylogenix, uses Manhattan

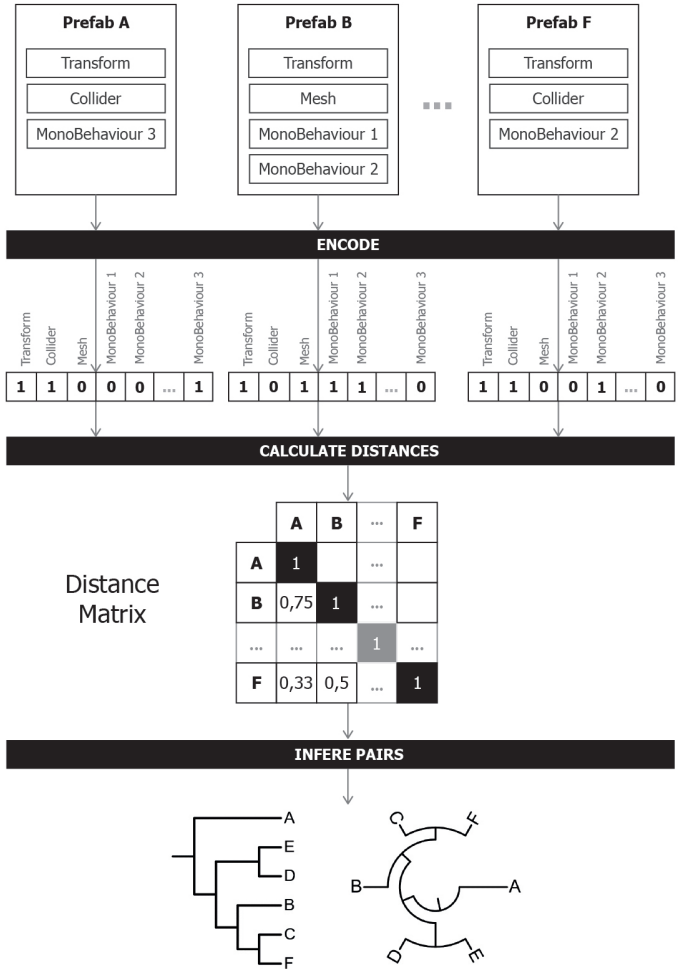

Figure 1: Representation of a phylogenetic analysis flow of a set of *Prefabs*

Distance, as it fits the characteristics of our encoding (same length in genomes), and enables to count the genes that are different in each genome [16]. With all of the distances, a distance matrix is generated, as shown in Figure 1.

Third, the Neighbor-Joinning Method is applied as inference technique to produce, from the distance matrix, the relationships of a phylogenetic tree. This inference method is widely used in biology [25].

Finally, as can be seen in Figure 1, a phylogenetic tree is generated according to the results obtained in the previous step.

Our tool, Phylogenix, is composed of two different parts: a Unity plugin, which analyzes *Prefabs*, realizes the phylogenetic inference, and generates the tree; and a web-based viewer that enables developers to visualize and explore the tree and export it as an image using the most common phylogenetic tree representations. The Unity plugin has been developed in `C#`, and it is distributed as a Unity package. It can analyze any Unity video game project. Figure 2 shows the main view of Phylogenix.

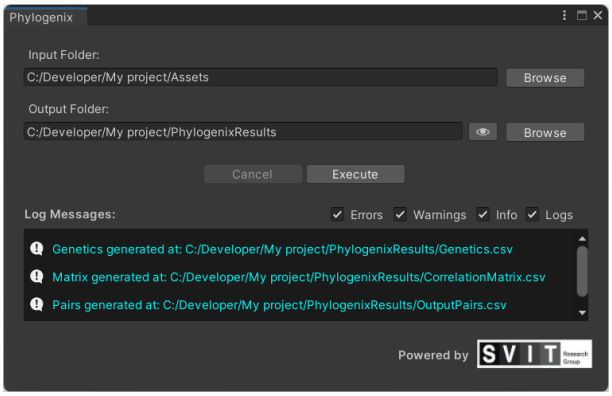

Figure 2: Phylogenix' panel included in the plugin

Table 1: Characteristics of case studies

	Num. *Prefabs*	Project size	Execution time
VTM	84	3,6 GB	1 minute
Grabitoons!	158	6,5 GB	2 minutes
Neon Hat	1319	11 GB	6 minutes
Toy Tactics	2036	134 GB	25 minutes

5 CASE STUDIES

We have validated our tool in a set of four commercial video games:

- **Vampire: The Masquerade (VTM)**[2]**:** It is a Computer Role-Playing Game in which the player rolls the dice, and takes decisions to control the story of the main character: a human with vampire powers.
- **Toy Tactics**[3]**:** Developed by Kraken Empire, Toy Tactics is a Real Time Strategy in which the player places their toys in the arena trying to defeat their enemies.
- **Neon Hat**[4]**:** It is a game designed to be played in virtual reality (VR) for PlayStation 4. Entalto Games created a world full of neon, where you must dodge and shoot, competing against others in global leaderboards. Neon Hat was awarded as Best game in VR of 2021 in the DeVuego Awards.
- **Grabitoons!**[5]**:** It is a physics-based galactic party game for up to four players, in which you must compete with your friends and beat them over the head.

Table 1 presents the characteristics of the games analyzed as case studies, and the time taken by Phylogenix to analyze each project.

Figure 3 shows the phylogenetic tree resultant of analyzing the Unity project of Grabitoons!. This phylogenetic tree shows a structured view of all of the *Prefabs* of the game, according to the genetic relations among them. For example, as can be seen in the zoomed zone, all `Canvas` *Prefabs* are in the same branch of the tree.

Additionally, this tree can be considered as a first step in the formalization of the variability process, as it provides an ordered and structured view of the *Prefabs* of the game. Also, it has been demonstrated that this tree can be relevant for latent content generation in a family of products [9].

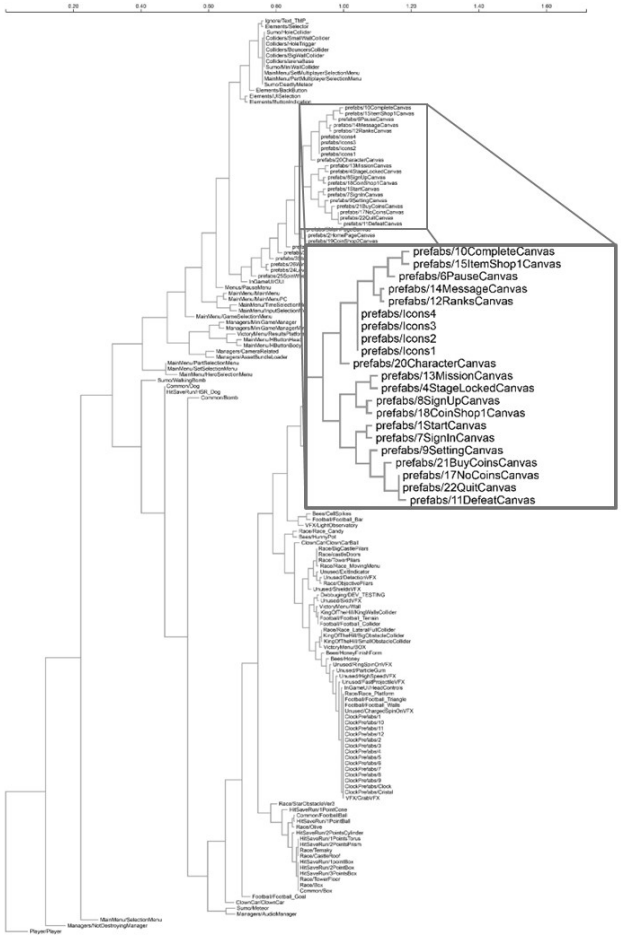

Figure 3: Phylogenetic tree from Grabitoons! video game

6 RELATED WORK

There are many different tools for Unity game engine focused on animations, intelligent systems, or simulation and visualization of terrains, environments, and systems [7, 13, 20, 26]. However, our focus is to analyze variability using phylogenetics.

Some other efforts in the literature apply SPLs to video games. Trasobares et al. [29] evaluated the benefits of SPLs, and Chueca et al. [10] compared SPL vs. Clone and Own in GSE. Castro and Werner [8] present a prototype of a game that was developed by applying a dynamic SPL to generate game modifications systematically. Lima et al. [18] present a work about Product Line Architecture recovery. Debbiche et al. [14] analyzed five Java games, and migrated three of these into a composition-based SPL implemented with *FeatureHouse* [2].

Focused on PCG, Preuss et al. [24] examine the interplay between quality and diversity in PCG for game development. Melotti et al. [19] introduce and implement the Deluged Novelty Search

[2]https://store.steampowered.com/app/1926120/Vampire_The_Masquerade__Heartless_Lullaby/
[3]https://store.steampowered.com/app/1772530/Toy_Tactics/
[4]https://store.steampowered.com/app/1908640/NeonHAT/
[5]https://store.steampowered.com/app/2208820/Grabitoons/

Local Competition algorithm (D-NSLC), which utilizes morphological niches to promote solution diversity in PCG for a game. Blasco et al. [5, 6] explored the application of Search-Based Software Engineering (SBSE) for content generation, using an evolutionary algorithm steered by simulations that incorporate the generated content. Neither of the works above-mentioned use phylogenetic inference for PCG as we are promoting with our tool.

Finally, Chueca et al. introduced in [9] a new approach for generating content for video games. They represent the relationships among the products of a family (Non-Playable Characters of a game) in a single tree. Developers use this information as a seed to create new Latent Content. However, our work presents a generic tool that can be applied to any video game project developed within the Unity game engine.

7 CONCLUSIONS

Phylogenix is an approach to introduce phylogenetics to GSE in Unity, providing developers with a tool that can be integrated in the environment in which their games are developed.

The tool analyzes the *Prefabs* of a video game project, performing a phylogenetic inference process, and generating a tree representation with the relationships extracted from the inference.

Moreover, the resulting tree can be used for generating new content, or as a first classification in the process of software variability formalization.

Future work is focused on uploading the plugin to the Unity Asset Store to make it available to anyone. We also aim to provide the tool with more phylogenetic inference methods used in biology. Improving the visualizer of the trees is planned too.

ACKNOWLEDGMENTS

This work was supported in part through the Spanish National R+D+i Plan and ERDF funds under the Projects VARNETICA (CNS2023-145422) and VARIATIVA (PID2021-128695OB-I00), and in part by the Gobierno de Aragón (Spain) (Research Group S05 20D).

REFERENCES

[1] Apostolos Ampatzoglou and Ioannis Stamelos. 2010. Software engineering research for computer games: A systematic review. *Information and Software Technology* 52, 9 (2010), 888–901.

[2] Sven Apel, Christian Kästner, and Christian Lengauer. 2011. Language-independent and automated software composition: The FeatureHouse experience. *IEEE Transactions on Software Engineering* 39, 1 (2011), 63–79.

[3] David A Baum and Stacey D Smith. 2012. Tree thinking: an introduction to phylogenetic biology. In *Tree thinking: An introduction to phylogenetic biology*. 476–476.

[4] David A Baum, Stacey DeWitt Smith, and Samuel SS Donovan. 2005. The tree-thinking challenge. *Science* 310, 5750 (2005), 979–980.

[5] Daniel Blasco, Jaime Font, Francisca Pérez, and Carlos Cetina. 2023. Procedural content improvement of game bosses with an evolutionary algorithm. *Multimedia Tools and Applications* 82, 7 (2023), 10277–10309.

[6] Daniel Blasco, Jaime Font, Mar Zamorano, and Carlos Cetina. 2021. An evolutionary approach for generating software models: The case of Kromaia in Game Software Engineering. *Journal of Systems and Software* 171 (2021), 110804.

[7] Andreas Brännström and Juan Carlos Nieves. 2022. A framework for developing interactive intelligent systems in unity. *Engineering Multi-Agent Systems (EMAS 2022)* (2022).

[8] Diego Castro and Cláudia Werner. 2021. Rebuilding games at runtime. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 73–77.

[9] Jorge Chueca, Daniel Blasco, Carlos Cetina, and Jaime Font. 2024. Leveraging Phylogenetics in Software Product Families: The Case of Latent Content Generation in Video Games. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference-Volume A*.

[10] Jorge Chueca, Jose Ignacio Trasobares, África Domingo, Lorena Arcega, Carlos Cetina, and Jaime Font. 2023. Comparing software product lines and Clone and Own for game software engineering under two paradigms: Model-driven development and code-driven development. *Journal of Systems and Software* 205 (2023), 111824.

[11] Jorge Chueca, Javier Verón, Jaime Font, Francisca Pérez, and Carlos Cetina. 2023. The consolidation of game software engineering: A systematic literature review of software engineering for industry-scale computer games. *Information and Software Technology* (2023), 107330.

[12] Charles Darwin. 1859. *On the origin of species by means of natural selection, or preservation of favoured races in the struggle for life*. John Murray.

[13] Emanuele De Pellegrin and R Petrick. 2023. PDSim: Planning Domain Simulation and Animation with the Unity Game Engine. In *ICAPS 2023 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

[14] Jamel Debbiche, Oskar Lignell, Jacob Krüger, and Thorsten Berger. 2019. Migrating Java-based apo-games into a composition-based software product line. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. 98–102.

[15] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1 (2013), 1–22.

[16] Eugene F Krause. 1986. *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation.

[17] Philippe Lemey, Marco Salemi, and Anne-Mieke Vandamme. 2009. *The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press.

[18] Crescencio Lima, Ivan do Carmo Machado, Eduardo Santana de Almeida, and Christina von Flach G. Chavez. 2018. Recovering the product line architecture of the Apo-Games. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*. 289–293.

[19] Alexandre Santos Melotti and Carlos Henrique Valerio de Moraes. 2018. Evolving roguelike dungeons with deluged novelty search local competition. *IEEE Transactions on Games* 11, 2 (2018), 173–182.

[20] Michal Pasternak, Nafiseh Kahani, Mojtaba Bagherzadeh, Juergen Dingel, and James R Cordy. 2018. Simgen: A tool for generating simulations and visualizations of embedded systems on the unity game engine. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 42–46.

[21] Georgios A Pavlopoulos, Theodoros G Soldatos, Adriano Barbosa-Silva, and Reinhard Schneider. 2010. A reference guide for tree analysis and visualization. *BioData mining* 3 (2010), 1–24.

[22] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. 2005. *Software product line engineering: foundations, principles, and techniques*. Vol. 1. Springer.

[23] Cristiano Politowski, Fabio Petrillo, João Eduardo Montandon, Marco Tulio Valente, and Yann-Gaël Guéhéneuc. 2021. Are game engines software frameworks? A three-perspective study. *Journal of Systems and Software* 171 (2021), 110846.

[24] Mike Preuss, Antonios Liapis, and Julian Togelius. 2014. Searching for good and diverse game levels. In *2014 IEEE Conference on Computational Intelligence and Games*. IEEE, 1–8.

[25] Naruya Saitou and Masatoshi Nei. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution* 4, 4 (1987), 406–425.

[26] Il-Sik Shin, Mohammadamin Beirami, Seok-Je Cho, and Yung-Ho Yu. 2015. Development of 3D terrain visualization for navigation simulation using a Unity 3D development tool. *Journal of Advanced Marine Engineering and Technology* 39, 5 (2015), 570–576.

[27] Gangavarapu Sivalaya, Bandaru Mounika, Gangavarapu Sailasya, and N Suresh Kumar. 2020. Implementation of Augmented Reality Application using Unity Engine Deprived of Prefab. (2020).

[28] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[29] Jose Ignacio Trasobares, África Domingo, Lorena Arcega, and Carlos Cetina. 2022. Evaluating the benefits of software product lines in game software engineering. In *Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume A*. 120–130.

[30] Grady Weyenberg and Ruriko Yoshida. 2015. Reconstructing the phylogeny: Computational methods. In *Algebraic and Discrete Mathematical methods for modern Biology*. Elsevier, 293–319.

[31] Tom Wijman. 2023. Global Games Market Report. https://newzoo.com/resources/trend-reports/newzoo-global-games-market-report-2023-free-version

A APPENDIX: PHYLOGENIX DEMONSTRATION

In this demonstration, the presenter introduces our tool Phylogenix, a Unity plugin to perform the phylogenetic analysis for a family of video game content where variability is not formalized. To contextualize the audience, the presenter explains that the video games industry has become the largest entertainment sector and that half of the software developers in the world are involved in the creation of video games. After that, the presenter introduces Game Software Engineering (GSE), as a recently emerged paradigm to address the complexity of software in the video games industry.

Next, game engines are presented, explaining their foundations and main characteristics. Unity engine is introduced as one of the most used game engines in the industry, and the concept of GameObject and *Prefab* are presented too. A simple example illustrates these concepts.

Later, the presenter explains that the creation of new content is one of the bottlenecks of video games industry. The presenter also introduces that some works from GSE community propose Procedural Content Generation techniques, but without formalizing the variability among the family. Moreover, the presenter introduces a recent work that has applied the concept of phylogenetics from biology to the industry of video games. The presenter explains that this approach can be used to create new content from an existing family, and also can be used as a form of Domain Analysis, as a first step towards formalization of the variability.

Next, an introduction to phylogenetics is realized, in order to explain the basic concepts of phylogenetics. The presenter introduces how our tool performs the phylogenetic analysis of a Unity project. First, the presenter explains how the *Prefabs* are analyzed to generate the genome of each one. Second, the presenter explains how the genomes are compared pair by pair, calculating the genetic distance (using Manhattan Distance), and producing a distance matrix. Third, the presenter explains how the pairs are joined using the Neighbor-Joinning method, Finally, the presenter explains how the pairs are represented in Newick format, and as trees in rectangular and circular representations.

The presenter provides an overview of the tool presented, composed of two parts: a Unity plugin, that makes the phylogenetic analysis, and a web-based viewer that enables to view and explore the trees generated by the plugin.

Then, the presenter opens the tool and makes a demonstration using a commercial video game, realizing the phylogenetic analysis using the plugin, and showing the results using the viewer.

The presenter concludes the tutorial by summarizing the idea behind Phylogenix, and how this tool can leverage the development process of video games, helping in latent content generation, as well as in the variability formalization of existing product families.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009