Journal of Architecture, Arts and Heritage (JAH) – Vol 3, Issue 1, February 2024
e-ISSN: 2822-437X

Environment Design in Medieval Themed Games

Emre Önal[1*]
Ankara Yildirim Beyazit University
emrek1@gmail.com
ORCID No: 0000-0001-6703-9143

Abdullah Bülbül[2]
Ankara Yildirim Beyazit University
mabulbul@aybu.edu.tr
ORCID No:0000-0002-2527-2729

Submission Date: 27.12.2023 / Acceptance Date: 04.02.2024

ABSTRACT
The medieval theme has long been a beloved and popular choice in entertainment, from novels to television shows and now video games. With the advent of computer technology, the middle ages have been simulated in virtual worlds, allowing people to experience its atmosphere in a more immersive and engaging way. As graphics hardware and software have advanced, these worlds have become even more fascinating and realistic.
This article discusses the process of designing medieval environments in 3D games, focusing on the key design elements of landscape and settlements. It explains how procedural techniques can be used to create large and immersive environments, and how modular design can be used to efficiently create a variety of scene objects. The article also highlights the importance of considering the gameplay and camera angle when designing environments, and how different game genres may require different design approaches. It is concluded by discussing the role of game engines in providing tools and features to support the creation of medieval environments, including terrain generation, city design, vegetation, architecture, and lighting. Overall, the article provides a comprehensive overview of the design process for creating engaging and immersive medieval game environments.

KEYWORDS
Game Environment Design, Medieval,3D Modeling, Game Development, Virtual Worlds

INTRODUCTION

The medieval period has long been a source of fascination and inspiration, with its rich history, architecture, and culture capturing the imagination of people across generations. This theme has been explored in various forms of media. Early on, novels introduced readers to this historic era. As visual mediums developed, television shows brought the medieval period to life on screens. With the advent of computer technology, the middle ages have been simulated in virtual worlds, allowing people to experience its atmosphere in a more immersive and engaging way (Romera et al., 2018; pp. 521). As graphics hardware and software tools have advanced, these worlds have become even more fascinating and realistic.

Over time, the craft of designing virtual environments for games matured into a specialized discipline. Distinct toolsets and best practices emerged to support building virtual worlds at large scales. In this article, we delve into the process of designing 3D environments for medieval games, focusing on the unique challenges and considerations that come with creating these virtual worlds. We explore the various techniques and tools used by game designers to bring these medieval settings to life.

Medieval themed games can be classified into 2D or 3D based on their game worlds, with 3D games offering a more immersive and interactive experience for players. In this article, we focus exclusively

[*]**Corresponding author.**

on 3D gaming environments, examining the various elements that contribute to their design and development.
Procedural content generation has revolutionized the production of virtual worlds by automating repetitive manual tasks. In games simulating medieval settings, procedural techniques are invaluable for efficiently authoring expansive landscapes, ecosystems, and settlements at scale. However, the field remains active as continued methodological advancements aim to optimize realism, performance, and design flexibility.

This paper aims to provide an overview and analysis of:

- Design concerns in all aspects of medieval environment design in games,
- Current procedural generation approaches employed for creating these environments,
- Use of these techniques within game engines.

In this direction, we catalog and describe the major procedural techniques used for terrain generation, vegetation simulation, architectural modeling, and other related tasks, together with an overview of the literature. We then synthesize how these methods are integrated into modern game engines to streamline workflows. To achieve these objectives, we benefit from the common practices in the industry in addition to the academic literature on the topic. Throughout the article, examples are provided from popular commercial games that illustrate the concepts described. Whether from the perspective of game designer, artist, or simply a fan of medieval games, this article provides valuable insights into the process of creating these captivating virtual worlds. The following sections will summarize the analyzed techniques and discuss the implications of the findings. A literature overview is provided together with the discussions.

DESIGN ELEMENTS

When designing a 3D medieval game environment, we can evaluate the design elements to consider under two main categories: landscape creation, and building and settlement design.

LANDSCAPE GENERATION

Unlike more modern or futuristic settings, natural landscapes play a prominent role in virtual medieval worlds. To achieve authenticity, these game environments typically feature expansive outdoor spaces with terrain features like towering mountains, winding rivers, and jagged cliffs. While 3D modeling programs allow designers to sculpt detailed terrain surfaces at the mesh level, manually authoring landscapes of such grandeur is simply impractical. To address this, over time developers have devised sophisticated computational techniques that simulate and streamline the generation of natural-looking terrain. Game engines then integrate these algorithms, combining procedures optimized for different generation tasks. The tools abstract complexity and provide designers with intuitive control over building complete landscapes efficiently. Fundamentally, there are distinct stages in the procedural process, each contributing to the final believable landforms players explore.

Heightmaps

A fundamental element in virtual terrain generation is the heightmap. In computer graphics, a heightmap stores numerical elevation data for a 3D landscape projected onto a 2D plane. The value at each coordinate point on the plane corresponds to the relative height of the terrain at that location. Heightmaps provide an efficient means of representing natural landforms.

Real-world elevations are irregular rather than uniformly sloped. Therefore, to construct believable in-game terrain, we need a method that generates somewhat random yet continuous height information. Procedural noise algorithms adeptly fulfill this role (Lagae et al., 2010; pp. 2579). There are various approaches but in its most commonly used form, the space is divided into hypercubic grids (squares in 2D) and random values are assigned to the lattice points. Then the values inside of each grid are calculated using the values of the surrounding lattice points. The lattice values are used in different ways. A basic procedural noise method called value noise smoothly interpolates these lattice values to calculate the interior of a grid. Smooth interpolation functions are used to calculate a smooth transition for inputs from 0 to 1 by having a gradient of zero at both 0 and 1. This provides a smooth transition at the intersections between the grids. In Figure 1, an example of these concepts is depicted on one dimension. Random values between -1 and 1 are assigned to lattice points, indicated with ticks, on a line. The values between the lattice points are constructed in three ways. It is apparent that a continuous heightmap cannot be achieved solely using the discrete values provided by the lattice points, as depicted on the left When the two surrounding lattice values are linearly interpolated, as shown in the center, the heightmap becomes continuous, but the transitions are sharp and each segment appears flat. Conversely, on the right, the lattice values are smoothly interpolated, resulting in a much more feasible heightmap for procedural terrain generation.

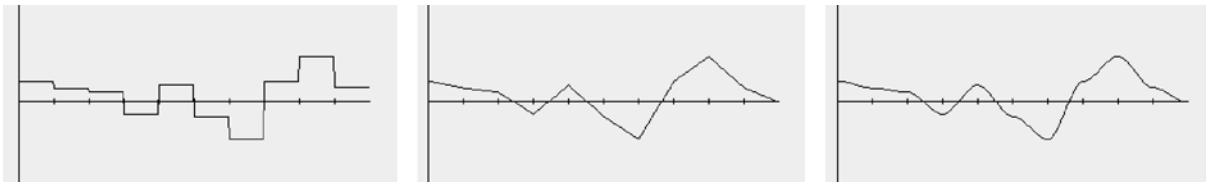

Figure 1. *1D heightmaps created using distinct random values(left), linear interpolation of the random values(center), and the value noise algorithm(right).*

Applying the same principles across 2D planes using techniques like Perlin noise (Perlin, 1985; pp. 287) or simplex noise (Perlin, 2001), (Gustavson, 2005) yields the type of irregular elevations that can be used to form a basis for a real terrain. These 2D heightmaps have become a standard element employed in the creation of terrains in games (Galin et al., 2019; pp. 553).

Perlin noise, a famous noise algorithm, has served as an industry standard in numerous procedural noise applications for decades. It is a gradient based algorithm. The lattice values are set to zero, or a certain mean value. Then, random gradients are assigned to lattice points. To calculate the value of a point within a grid, extrapolations of the surrounding gradients at that point are interpolated smoothly. Ken Perlin later made performance and visual improvements to his algorithm (Perlin, 2002; pp. 681). He also developed another algorithm called simplex noise that overcomes the weaknesses of Perlin noise. In simplex noise, the space is divided into n-dimensional simplices (triangles in 2D) instead of hypercubic grids. And it uses radial summation instead of sequential interpolations along each dimension. Although research that compare and evaluate different techniques along with simplex noise (Archer, 2011; pp. 378), (Rose &Bakaoukas, 2016; pp. 1), (McEwan et al, 2012; pp. 85), (Hyttinen, 2017), highlight it as superior to Perlin noise and consider it a potential substitute, the industry has not notably shifted in that direction. The complexity of its implementation and the fact that Perlin noise is deemed sufficient for many scenarios could be reasons behind this. Interestingly, there also appears to be limited detailed academic discussion of simplex noise available despite its known advantages.

Figure 2 presents an example of two-dimensional heightmap. It is created using a noise algorithm we developed by combining different features of Perlin and simplex noise algorithms. It uses gradient

interpolation on a simplex grid structure. In the figure, the elevations are distinguished by a basic color gradient. This low-cost approach provides a preliminary visual representation of the terrain's expected biomes across different altitudes. Further decoration methods, which are described in the vegetation section below, enhance the authenticity of the generated landforms.

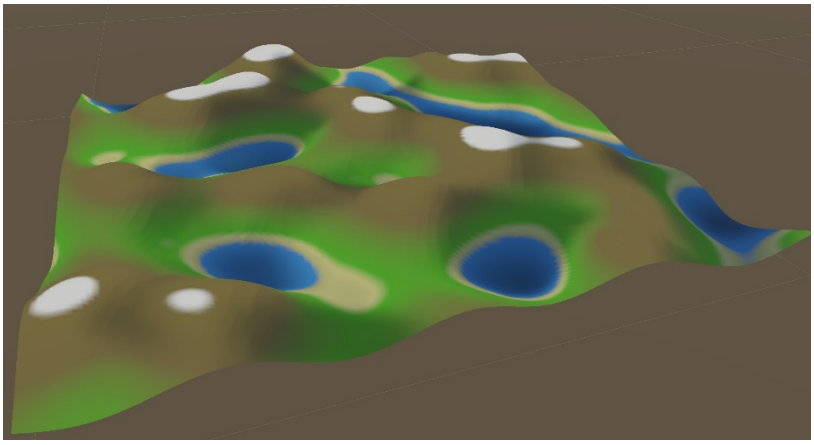

Figure 2. *2D heightmap created using interpolated simplex noise algorithm. Colors change as blue, yellow, green, brown, and white to represent the characteristics of the surface at different elevations.*

Surface Details

Basic noise algorithms create soft shapes as in Figure 2. Adding multiple noisemaps at different scales on top of each other helps the final heightmap to have finer details as in the left image in Figure 3, however, it is important to consider other factors in nature that play a role at shaping terrains. To improve the reality, physical simulations are used to model the natural phenomena that impact the surfaces of real landscapes(Musgrave et al., 1989; pp. 41), (Beneš et al., 2006; pp. 99), (Chiba et al., 1998; pp. 185). These are mostly different types of erosion effects, which are modelled by the idea of moving sand particles over the terrain surfaces from one place to another by gravity, wind or rain. After obtaining base terrains with noise algorithms, these erosion algorithms are applied to improve surface features of terrains (Galin et al., 2019; pp. 553). The image on the right in Figure 3 more closely resembles the terrain images one might observe on platforms like Google Earth.

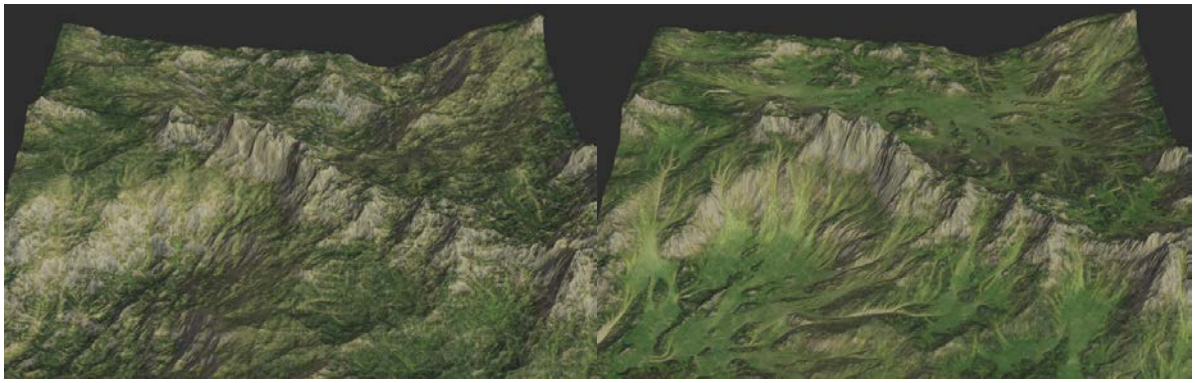

Figure 3. *Terrain images created by a program called "World Machine" (URL-14). Left is the image of a terrain created using Perlin Noise. After applying the erosion algorithm, the image on the right is created.*

Vegetation

After having the heightmap, the subsequent step is to create the elements that cover the surface of the terrain. Manual crafting may be infeasible in this process as well. Instead, procedural techniques are employed, drawing inspiration from various concepts found in nature (Cordonnier et al., 2017; pp.

1), (Ketabchi et al., 2015; pp. 98). For instance, surfaces with minimal slopes are often populated with grass and trees, while higher elevations experience vegetation loss and become blanketed in snow.

In earlier games, which were designed for computers with limited graphics power, different surface features were represented by 2D textures of related materials like grass, rock, snow, etc. Two examples are shown in Figure 4.

Figure 4. *Image from Warcraft III (left) (URL-1), image from Fable Anniversary (right) (URL-2).*

On the other hand, with graphics cards becoming more powerful, the use of much more realistic 3D objects for foliage and other small details became possible. Figure 5 showcases a scene created in Unreal Engine where the ground is covered with many detailed 3D objects. The image is not an illustration, or a rendered static image. It is taken from an interactive simulation.

Figure 5. *Image taken from the "Medieval Game Environment" scene created in Unreal Engine (URL-3).*

In the realm of vegetation generation, an interesting approach is the use of L-systems (Prusinkiewicz & Lindenmayer, 2012). L-systems, short for Lindenmayer systems, are a type of formal grammar that can be used to model the growth and development of plants. They provide a way to describe the rules and processes that govern the branching patterns and shapes of plants.

L-systems consist of a set of symbols, called the alphabet, and a set of production rules that define how the symbols can be replaced or expanded. By iteratively applying these production rules, complex and intricate plant structures can be generated. With L-systems, a single character or symbol can represent a specific action, such as branching or elongation, and the production rules determine how these actions are carried out. By combining different symbols and rules, a wide variety of plant forms can be

simulated, ranging from simple trees to intricate foliage. They enable the generation of diverse and realistic vegetation in virtual environments in a procedural manner.

BUILDING AND SETTLEMENT DESIGN

Camera Angle & Environment Detail

Camera angle is a decisive factor on how to design environmental elements in a game (Laurier & Reeves, 2014; pp. 181). It changes according to gameplay requirements. In strategy games, an overview of the map is the focus of attention throughout the game. So, top-down view is the common camera perspective in strategy games (Fig. 6). Usually, the structures do not need to be modeled in high detail in these games.

Figure 6. *Strategy game: Age of Empires III (URL-4).*

City building games became popular in the last decade. In these types of games the camera concentrates on a smaller region on the map but top-down view is still the preferred angle most of the time (Fig. 7). In some cases, city building games may also incorporate elements of survival and crafting (Fig. 8), with third-person or first-person camera views that allow players to interact with smaller equipment and objects.

Figure 7. *City Building Games: Manor Lords (left) (URL-5), Terrascape (right) (URL-6).*

Figure 8. *Survival/Crafting Games: Medieval Dynasty (left) (URL-7), Life is Feudal (right) (URL-8).*

In role-playing games (RPGs) (Fig. 9), players interact with the game world from a close-up perspective using a third or first-person camera view. This requires a higher level of detail not only for the environment but also for the interiors of buildings, as players may enter and explore them. Therefore, building and settlement design in RPGs often involves creating detailed 3D models of both the exterior and interior of structures.

Figure 9. *Action Role Playing Game: Kingdom Come Deliverance (URL-9).*

Placing Settlements on Landscapes

In today's urban environments, we often reshape the land to suit our needs and build settlements using modern technology. However, in medieval games, the common scenery is huts, towns, or castles set amidst vast landscapes (Fig. 7, Fig. 9). Creating these settlements within a wide landscape can be a massive workload, just like manual landscape creation. Procedural methods used in landscape creation have matured to the point where manual intervention is no longer necessary. However, while some games use procedural techniques for creating settlements, there is no globally accepted standard method in the industry. There are a few academic studies on the procedural placement of buildings on landscapes (Zmugg et al., 2014; pp. 1009), and on the creation of whole towns (Emilien et al., 2012; pp. 809), (Bulbul, 2023).

Modular Design

In strategy games, the buildings are often seen only from the outside and the same models are used throughout the game. However, in other types of games such as RPGs, modeling large cities and interior spaces of buildings may be required for gameplay. Creating unique models for every building would be a massive workload and would require a large team of 3D artists. Moreover, it would be challenging to maintain consistency across all the buildings. To address this issue, many games use a technique called modular design (Statham et al., 2022). In modular design, a themed kit is created first, which consists

of a set of standardized architectural components - such as wall sections, doors, windows, and rooflines - designed to be combined to create larger structures. This approach allows developers to quickly create a variety of structures while maintaining consistency throughout the game. In Figure 10 there is an example of a modular kit on the left and a scene created with the pieces from the kit on the right.

Figure 10. *An environment asset pack from Unreal Engine Marketplace by Hivemind. A collection of model parts and a scene created with these parts (URL-10).*

The available literature on the subject of modular design is relatively scarce. Most of the existing content is shared through conference talks or advice from industry experts (Burgess &Purkeypile, 2016), (Perry, 2002; pp. 30). While there is no standardized method, the key considerations in designing a modular kit can be summarized as follows:

- *Flexibility:* The modular assets should be designed to fit well with other related assets in the kit. And they should allow for the creation of various meaningful combinations.
- *Art style:* Consistency in the design style of the assets is crucial for providing visual coherence in the scenes created from the kit (Johansson, 2017).
- *Diversity:* While maintaining a consistent art style, the kit should offer a sufficient range of asset variations to avoid visual repetition.
- *Metric features:* It is beneficial to establish a grid size and use multiples of it as the size of the assets. Another beneficial practice for modularity is to use consistent pivot points for the assets. Game engines allow snapping objects to invisible grids when while moving them. So, these practices make the level designer's job easier when combining the parts together. This becomes even more crucial when the assets in the kit are programmatically utilized to construct larger structures or populate environments in a procedural manner.

In summary by generating a landscape procedurally and then filling it with efficient use of a well designed modular kit, it is possible to obtain large immersive environments even for smaller sized development teams.

Figure 11. *Turn based strategy game: Civilization VI (URL-11).*

Style Preference

While realism is something appreciated in game environment design, it is not always the primary goal. Designers may opt for a more stylized or cartoonish aesthetic to create a unique and engaging atmosphere. For example, the elements in Warcraft III (URL-1) (Fig. 4) are far from realistic, yet they provide an attractive fantastic ambiance. In other cases, games that prioritize data management may choose to use simplified, iconic representations of in-game elements to improve gameplay usability. For instance, in the turn-based strategy game Civilization VI (URL-11) (Fig. 11), units, cities, wonders, and improvements are all designed to fit within a hexagonal tile-based map. Another similar example, depicted in Figure 12, showcases a game map divided into clickable hex tiles, each offering distinct functionalities. On the left, we observe the wireframe model of the data stripped from the game's thematic elements. On the right, the wireframe data is augmented with elements from a medieval setting, transforming the visual representation accordingly. It's worth noting that the same data model can be visualized with a different theme, such as a space setting where resource tiles correspond to planets, and players collect resources through small spaceships, comparable to how players in the medieval setting gather resources from trees and rocks through townsmen. Ultimately, the choice of environment design style and approach will depend on the goals and priorities of the game, as well as the preferences of the target audience.

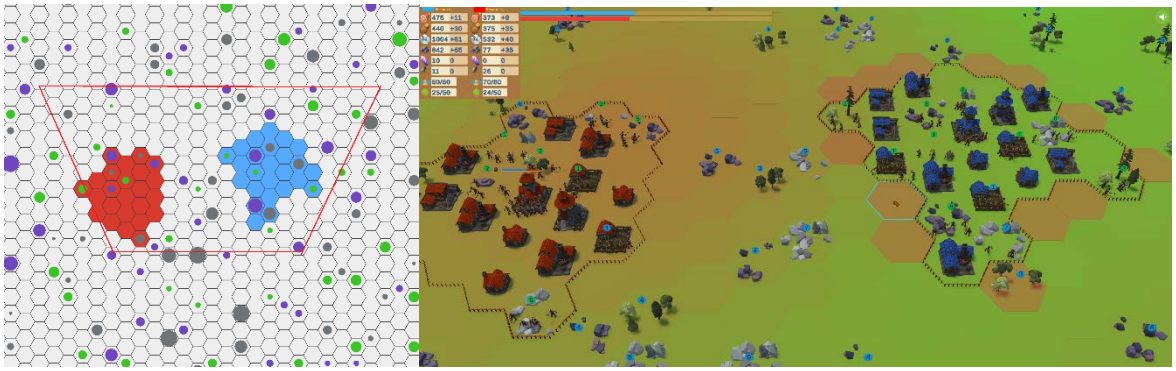

Figure 12. *On the left, wireframe data representation of game map. The discs and their size represent resources and their productivity level. On the right, visualization of data with game elements from a medieval setting.*

GAME ENGINE

To create a game environment, 3D models are often crafted using software like Maya or Blender, which allow artists to manipulate polygons and create detailed models. However, manually creating an entire game environment in this way is time-consuming and impractical. This is where game engines come in. Game engines are software platforms that provide designers with the tools they need to build out their levels and game worlds. Various software options offer different features (Christopoulou & Xinogalos, 2017). Unity3D and Unreal Engine (Fig. 13) (URL-12,13) are two popular examples of game engines that support the features we've discussed and offer user-friendly interfaces for designers to work with.

Once 3D models are created, they can be imported into the game engine as reusable assets. These assets, which can include materials, textures, and animations, are stored in resource files and can be accessed by the designer in a scene editor, where they can view, navigate, and prototype their world in real-time with an easy to control drag-and-drop user interface. Designers can either create these assets in-house or obtain them from external sources. Unity3D and Unreal Engine both have their own marketplaces filled with assets, tools, and plugins created by independent artists and programmers, making it easy to find the resources needed to bring a game environment to life.

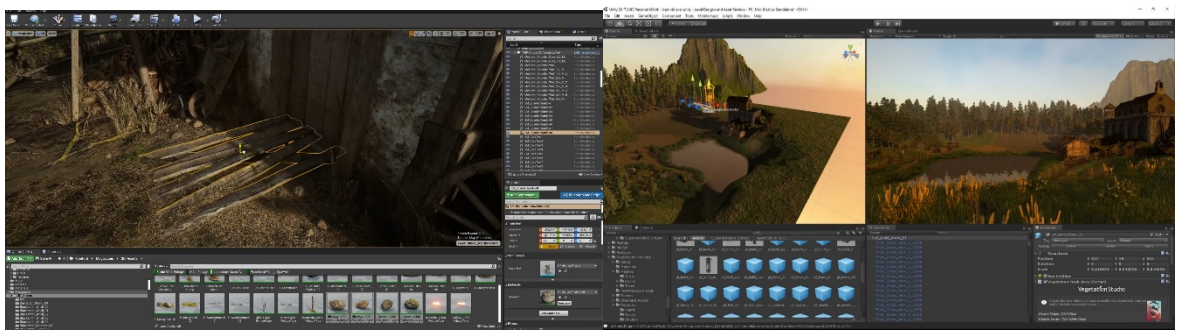

Figure 13. *Game engines: Unreal Engine (left) (URL-12), Unity3D(right) (URL-13).*

Game engines offer a wide range of tools and features to help designers create immersive medieval environments. Here are some key areas that engines support:

- Terrain generation: Game engines can generate fully procedural terrains, or allow designers to create terrains manually from scratch. Sculpting tools enable designers to shape landscapes like forests, hills, mountains, and valleys. Brush tools can be used to paint the surface of the terrain with different textures such as grass, rock, sand, and pebbles.
- City/Settlement design: Designers can drag and drop prefab assets onto the scene view to populate the landscape with castles, villages, farms, and other habitations. Scene view allows for easy movement and transformation of 3D models on the scene.
- Vegetation/Flora: The terrain system within game engines provides a basic vegetation view using textures. However, for more detailed vegetation elements, separate 3D models may be required, especially when designing locations that are close to the camera. Game engines often provide a variety of tree, grass, bush, and foliage models as built-in assets. Designers can also utilize custom models to achieve higher quality or specific artistic styles. To efficiently populate terrains with desired flora, high-end game engines offer brush tools. Some engines even support the procedural generation of forest layouts and groups of flowers.
- Architecture objects: Pre-made assets for buildings, siege weapons, wells, etc. can be customized and combined and created more if needed with the modular design concept we described.

- Lighting systems: Different lighting systems are implemented in game engines. Dynamic lighting can be used to illuminate the environment dynamically according to daytime. A point light used to create the light effect coming from a torch or a campfire can create a certain atmospheric effect in a scene. Particle illumination systems can be used on magical items.

With minor differences, prefabrication systems in engines allow designers to manage reusable assets efficiently when creating levels and game environments. In Unity3D, prefabs are a powerful tool for creating modular assets. They are essentially pre-configured game objects that can be reused throughout a project. Prefabs can include not only visual assets but also scripts, colliders, and other game logic. When a prefab is modified, it is possible to update all instances of that prefab in the scene, making it easy to maintain consistency across multiple assets. Prefabs can also be nested, allowing for complex modular structures to be created from smaller components. Prefab variants and material swapping enable visual diversification while maintaining shared rigging. For example, a wall prefab could have stone, wood, or brick material variants applied programmatically based on biome. Unreal Engine also provides robust support for modular design through its blueprint system. It offers a visual scripting approach to game development, making it ideal for modular design. Developers can create modular blueprints that encapsulate specific gameplay mechanics or interactions. These modular blueprints can then be easily reused and connected together to build complex systems. The drag-and-drop nature of blueprint scripting simplifies the process of assembling modular components, enabling designers to quickly iterate and experiment with different combinations of functionality.

Both Unity3D and Unreal Engine provide level streaming functionality, which is particularly beneficial for modular design. Level streaming allows developers to divide large game worlds into smaller, modular chunks known as levels. These levels can be loaded and unloaded dynamically, based on the player's position or specific triggers. By breaking down the game world into modular levels, developers can focus on designing and optimizing smaller sections at a time, improving performance and enabling efficient iteration. This approach also facilitates collaboration, as different team members can work on separate levels simultaneously.

CONCLUSION

Medieval theme is one of the most popular game environment concepts with its rich and immersive ambiance. Examples of many popular titles that leverage the appeal of this theme are showcased throughout the article. However, the creation of these games poses serious challenges for development teams since constructing a complete medieval environment is a compound process with its diverse set of artistic and technical design concerns. In this study, different aspects of medieval environment design in games, including both artistic concerns and technical requirements, are presented in a concentrated manner. The current state of procedural generation and modular environmental design approaches supporting the production of immersive medieval simulations within game engines is presented. A systematic analysis of relevant literature mapped the major techniques and discussed their integration. This article provides a comprehensive understanding of the process of creating these types of virtual worlds and serves as a valuable resource for artists, level designers, and programmers.

The creation of immersive and visually captivating medieval environments in games requires a careful balance of artistic design and technical implementation. Game engines offer a range of features specifically tailored for building these virtual worlds. Terrain generation tools enable designers to sculpt and paint intricate landscapes, while the inclusion of carefully crafted architecture assets, such as castles and villages, adds depth and authenticity to the medieval setting. The use of vegetation and

flora is also crucial in enhancing the overall ambiance of the environment. Game engines provide a combination of built-in and customizable vegetation assets, allowing for meticulous attention to detail.

Moreover, the modular design concept proves invaluable in streamlining the creation process, allowing for the customization and combination of architecture assets as needed. This approach not only saves time and resources but also ensures consistency throughout the game world. Procedural techniques and modular design methods help to ease the development process, and with advancements in machine learning, we are beginning to see their use in automating these techniques to create environments more efficiently.

Promising avenues for future research include enhanced machine learning applications. Procedural pipelines may benefit from trained models automating optimization or variation tasks. Biological distribution simulations could also evolve through adaptive systems.

Overall, by strategically blending human creativity and algorithmic automation, the procedural approach transforms development logistics. The expansion of unified rule systems, combined with growing engine support, will push medieval world-building capabilities far beyond individual manual efforts.

In conclusion, the creation of medieval game environments requires a comprehensive understanding of both artistic and technical elements. By leveraging the features and tools provided by game engines, designers can bring these worlds to life, captivating players with their immersive and visually stunning experiences. As technology continues to advance, the integration of procedural techniques and the exploration of new automated approaches hold promise for even greater ease and efficiency in the development of medieval game environments.

REFERENCES

Archer, T. (2011, April). Procedurally generating terrain. In *44th annual midwest instruction and computing symposium, Duluth* (pp. 378-393).

Beneš, B., Těšínský, V., Hornyš, J., & Bhatia, S. K. (2006). Hydraulic erosion. *Computer Animation and Virtual Worlds*, *17*(2), 99-108.

Bulbul, A. (2023). Procedural generation of semantically plausible small-scale towns. *Graphical Models*, *126*, 101170.

Burgess, J., Purkeypile, N. (2016). “Fallout 4’s” Modular Level Design. In Game Dev. Conf. GDC 2016, InformaTech, San Francisco, CA, US.

Chiba, N., Muraoka, K., & Fujita, K. (1998). An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and Computer Animation*, *9*(4), 185-194.

Christopoulou, E., &Xinogalos, S. (2017). Overview and comparative analysis of game engines for desktop and mobile devices.

Cordonnier, G., Galin, E., Gain, J., Benes, B., Guérin, E., Peytavie, A., &Cani, M. P. (2017). Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics (TOG)*, *36*(4), 1-12.

Emilien, A., Bernhardt, A., Peytavie, A., Cani, M. P., &Galin, E. (2012). Procedural generation of villages on arbitrary terrains. *The Visual Computer*, *28*, 809-818.

Galin, E., Guérin, E., Peytavie, A., Cordonnier, G., Cani, M. P., Benes, B., & Gain, J. (2019, May). A review of digital terrain modeling. In *Computer Graphics Forum* (Vol. 38, No. 2, pp. 553-577).

Gustavson, S. (2005). Simplex noise demystified. *Linköping University, Linköping, Sweden, Research Report*.
Hyttinen, T. (2017). *Terrain synthesis using noise* (Master's thesis).

Johansson, I. (2017). Defining what is visually dynamic for modular assets in level design.

Ketabchi, K., Runions, A., &Samavati, F. F. (2015, October). 3D Maquetter: Sketch-based 3D content modeling for digital Earth. In *2015 International Conference on Cyberworlds (CW)* (pp. 98-106). IEEE.

Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D. S., ... & Zwicker, M. (2010, December). A survey of procedural noise functions. In *Computer Graphics Forum* (Vol. 29, No. 8, pp. 2579-2600). Oxford, UK: Blackwell Publishing Ltd.

Laurier, E., & Reeves, S. (2014). Cameras in video games: Comparing play in Counter-Strike and the Doctor Who Adventures. *Studies of video practices: Video at work*, 181-207.

McEwan, I., Sheets, D., Richardson, M., &Gustavson, S. (2012). Efficient computational noise in GLSL. *Journal of Graphics Tools*, *16*(2), 85-94.

Musgrave, F. K., Kolb, C. E., & Mace, R. S. (1989). The synthesis and rendering of eroded fractal terrains. *ACM Siggraph Computer Graphics*, *23*(3), 41-50.

Perlin, K. (1985). An image synthesizer. *ACM Siggraph Computer Graphics*, *19*(3), 287-296.

Perlin, K. (2001). Noise hardware. In Real-Time Shading SIGGRAPH Course Notes. *Real-Time Shading SIGGRAPH Course Notes*, 23.

Perlin, K. (2002, July). Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (pp. 681-682).

Perry, L. (2002). Modular level and component design, Game Dev. (pp. 30–35).

Prusinkiewicz, P., &Lindenmayer, A. (2012). *The algorithmic beauty of plants*. Springer Science & Business Media.

Rose, T. J., &Bakaoukas, A. G. (2016, September). Algorithms and approaches for procedural terrain generation-a brief review of current techniques. In *2016 8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)* (pp. 1-2). IEEE.

San Nicolás Romera, C., Nicolás Ojeda, M. Á., & Ros Velasco, J. (2018). Video Games Set in the Middle Ages: Time Spans, Plots, and Genres. *Games and Culture*, *13*(5), 521-542.

Statham, N., Jacob, J., &Fridenfalk, M. (2022). Game environment art with modular architecture. *Entertainment Computing*, *41*, 100476.

Zmugg, R., Thaller, W., Krispel, U., Edelsbrunner, J., Havemann, S., &Fellner, D. W. (2014). Procedural architecture using deformation-aware split grammars. *The Visual Computer*, *30*, 1009-1019.

URL-1. Blizzard Entertainment(2002, June 3). *Warcraft III: Reign of Chaos*. Retrieved December 27, 2023, from https://lutris.net/games/warcraft-iii-reign-of-chaos/

URL-2. Big Blue Box Studios (2004, September). *Fable Anniversary's User Interface Breakdown*. Retrieved December 27, 2023, from https://gamingbolt.com/fable-anniversarys-user-interface-breakdown

URL-3. QuixelMegascans(2002, June 3). *Medieval Game Environment*. Retrieved December 27, 2023, from https://www.unrealengine.com/marketplace/en-US/product/scarecrow

URL-4. Tantalus Media and Forgotten Empires(2020, February). *Age of Empires III: Definitive Edition*. Retrieved December 27, 2023, from https://www.ageofempires.com/games/aoeiiide/

URL-5. Slavic Magic. *Manor Lords*. Retrieved December 27, 2023, from https://manorlords.com/

URL-6. Bitfall Studios(2023, April). *Terrascape*. Retrieved December 27, 2023, from https://store.steampowered.com/app/2290000/TerraScape/

URL-7. Render Cube(2002, June 3). *Medieval Dynasty*. Retrieved December 27, 2023, from https://www.xbox.com/tr-TR/games/store/medieval-dynasty/9pddp6ml6xhf

URL-8. Mindillusion(2016, August). *Life is Feudal: Forest Village*. Retrieved December 27, 2023, from https://lifeisfeudal.com/

URL-9. Warhorse Studios (2018, February 13). *Kingdom Come Deliverance*. Retrieved December 27, 2023, from https://www.kingdomcomerpg.com/

URL-10. Hivemind(2023, April 13). *Modular Medieval Town – Medieval Town*. Retrieved December 27, 2023, from https://www.unrealengine.com/marketplace/en-US/product/modular-medieval-town-megapack-medieval-town

URL-11. Firaxis Games(2016, October 21). *Civilization VI*. Retrieved December 27, 2023, from https://civilization.com/

URL-12. Epic Games(1998). *Unreal Engine*. Retrieved December 27, 2023, from https://www.unrealengine.com/en-US

URL-13. Unity Technologies(2005, June 8). *Unity Real-Time Development Platform*. Retrieved December 27, 2023, from https://unity.com/

URL-14. World Machine Software, LLC (2006, March). *World Machine: The Premier 3D Terrain Generator*. Retrieved December 27, 2023, from https://unity.com/