INTERNATIONAL JOURNAL OF EDUCATION AND INFORMATION TECHNOLOGIES
DOI: 10.46300/9109.2024.18.3 Volume 18, 2024

E-polis: An Innovative and Fun Way to Gamify Sociological Research with an Educational Serious Game – Game Development Middleware Approach

Alexandros Gazis[1,*], Eleftheria Katsiri[1,2]
[1]Department of Electrical and Computer, School of Engineering Engineering,
Democritus University of Thrace,
Xanthi, 67100,
GREECE

[2]Athena Research & Innovation Center in
Information Communication & Knowledge Technologies,
Institute for the Management of Information Systems,
Marousi, 15125,
GREECE

Received: March 7, 2023. Revised: February 6, 2024. Accepted: March 9, 2024. Published: April 17, 2024.

Abstract—E-polis is a serious educational game that uses gamification to study young people's opinions about their ideal society. Our game consists of a digital city where players navigate to complete quests. The answer to each quest changes the shape of the buildings and thus, after completing the core quests, the game world will create a unique neighborhood. In this way, we visualize the consequences of players' actions and simulate how their answers impact our urban society and environment. As such, E-polis can be used as a tool that based on some predefined inputs (quests/questions) can evaluate and log young people's views on the ideal society. Our serious game uses the Unity Game engine, and it can be used in various socio-economic case studies such as exploring civic engagement and social justice. Specifically, our article reviews the literature on game engines and defines how an educational serious game can gamify a learning experience. Then, we present in detail the software architecture design principles of our game and suggest a new generic middleware for digital game developers. Moreover, we expand on a new transition mechanism in Unity to re-use graphics dynamically using some pre-render real-time scene game objects. Our mechanism allows for better execution times of digital games in less resource-intensive computer systems when players transition through different scenes. The authors suggest that the technical novelty of this game lies in its middleware software entities' separation of concerns and this transition mechanism as a way to assist in game development and deployment. This is because it can provide game developers with several pre-built services and a generic scene transition mechanism, thus helping game developers to be more productive, reduce their costs, and improve the quality of their games.

Keywords—Gamification, Unity Game Engine, Game Engine Literature Review, Scene Transition Mechanism, Serious Game Development Middleware, Training, Virtual Reality, Learning through games, Educational Games, Serious Game UX, Game Development Platform, Open Source Game Development, Game Development for Education, Game Development for Training, Serious Games for Social Impact, Virtual Reality Learning, Educational Art Games, Novel Serious Game.

I. Introduction

MOORE'S Law, [1], suggests that the price of electronic computing devices is decreasing, while their capabilities, especially integrated circuits, double approximately every two years. Though an observation rather than a law, this statement has held since 1975 and is widely used as a measure of rapid technological advancement. The current era necessitates a digital transformation of businesses and everyday lives. Big Data and the Internet of Things are crucial terms showcasing how machine-to-machine communication creates a vast network of devices. These devices, empowered by increased hardware and software capabilities, generate live data, creating the need to analyze and understand this rapid information flow, [2].

Our lives are filled with devices like smartphones, laptops,

tablets, and phablets that can now handle intensive computing tasks demanding heavy CPU and RAM loads. Regardless of whether it's a mini-computer or a cheap phone, these devices typically have multiple threads and cores with at least 2-4 GB of RAM, eliminating the need for a high-voltage power supply, [3], [4]. This has led to the development of smart, low-cost, and low-power devices for more diverse applications, [5]. More recently, these advancements have been applied in chatbots, artificial intelligence (AI), multimedia content, and gaming, [6]. These capabilities have made digital games more accessible, allowing a larger audience to play locally or online without needing a high-end computer, [7], [8], [9], [10], [11].

In the last decade, graphics optimization has accelerated due to two key factors:

- Middleware shift: The transition from monolithic middleware to a software-oriented architecture allows resource-intensive processes to be shifted to the cloud or utilize machine-specific programming languages for optimal performance, [11], [12].
- Accessibility: This approach has enabled the development of beautiful and complex 2D and 3D graphics for a broader audience.

The term "digital games" encompasses all games, regardless of graphical fidelity. As long as two or more players can interact through multiplayer or networking features, these games offer an interesting case study on player interaction and how beliefs and opinions are shaped within the digital world, [13], [14]. This has led to the emergence of a new field of study focused not on the games themselves, but on the players and their learning experiences. Pedagogical games, a subgenre of digital games, prioritize education and training over pure entertainment, [14]. While prioritizing education, serious games (SGs) can still be fun and engaging. Their design emphasizes learning objectives but doesn't eliminate entertainment.

This article focuses on SGs and proposes a new game to study young people's behavior and how interaction might change their perspectives. The term SG emerged in the 1990s with the development of "Brown Box," a game to educate soldiers on radar use. Since then, SGs have been adopted in various fields, including the armed forces, legal systems, education, environmental concerns, social issues, and most notably, medical services, [15].

Our article introduces a novel SG for sociological purposes. This game simulates a digital city representing an ideal society. As players navigate, they encounter quests without predetermined right or wrong answers. Based on their beliefs, players choose how to interact with these dilemmas from a set of pre-defined questions and answers. After each choice, players receive feedback summarizing their decisions, and the city's shape, or even the entire city, may change to reflect the players' democratic choices (actions). Our game design is based on MIT's design architecture principles, incorporating recent approaches in serious games, such as the separation of concerns within the game's hierarchy and an emphasis on real-world issues for impactful purposes.

Our goal extends beyond simply presenting the game. We aim to explain its technical and theoretical contributions. Analytically, this game is technically interesting because it features a multi-layered cloud computing middleware that utilizes various processes and expands upon the capabilities of Unity's game engine. Additionally, since studying player behavior is the primary focus, the middleware's data layer stores player answers in both text files and local datasets. Notably, we have strived to implement simple and resource-efficient methods for data aggregation and storage using CSV/JSON and SQLite datasets.

On the theoretical side, the game proposes a novel framework for promoting empathy and understanding of real-world issues and their impactful consequences. Targeting a younger audience, the game seeks to explore how they envision and dream of an ideal society.

Specifically, in the following sections, we present our research on digital serious games, i.e., on games that have educational or social purposes. We then define serious games and explain the reason they are beneficial for various domains. We also outline our research objectives and questions as follows:

1. We review the existing literature on serious games, and how they have been classified and described by different authors. We also examine the platformer genre of digital games which involve jumping and running across platforms. We discuss how platformers can be adapted to create serious games that are fun and instructive.
2. We describe the innovative features of our approach, which integrates elements of serious games and platformers to create a game that is both entertaining and educational. We explain how our game combines gamification and simulation to study young people’s opinions about their ideal society.
3. We explain the technical aspects of our game development process, such as the choice of Unity as our game engine, the design of the Unity scenes, and the data collection techniques. We also introduce a novel scene transition mechanism that renders graphics dynamically, by using pre-rendered and real-time scenes. This mechanism enables smooth and seamless transitions between scenes.
4. Our conclusion summarizes key findings and proposes future research directions for fellow scientists in the field. Here's a breakdown of our contributions:
    a. Research Problem and Differentiation: We clearly define the research problem and differentiate our game from existing sociological serious games (SGs).
    b. Technical and Theoretical Contributions: We present the game's technical and theoretical contributions, focusing on the proposed game scene mechanics.

c. Literature Review and Engine Selection: We provide a detailed literature review of industry game engines used for educational digital game development, offering an in-depth technical comparison of their respective features.
d. Algorithmic Design and Implementation: We elaborate on the algorithm we developed in Unity, including the rendering and scene transition mechanisms we propose. We also discuss the chosen game engine and development structure used for our game.
e. Data Acquisition and Middleware Framework: We describe the acquired data and endpoints of interest during gameplay. We showcase our middleware framework, encompassing the platform, engine, game, and application layers, and propose a new middleware with a generic content development framework (applicable to both sociological and non-sociological genres) with adaptable front-end and back-end layers.

II. THE RESEARCH PROBLEM

The digital game developed in this project falls under the category of serious games (SGs) as it extends beyond mere entertainment. Its design principles delve into players' behavior and choices, educating them on how their actions can shape society. This SG, therefore, serves as a modern research tool with broad applicability in social sciences and education.

Specifically, this SG seeks to understand the characteristics of young individuals through a combination of quantitative and qualitative design methods. More specifically, we have created a virtual city where the player is presented with a preset number of quests. These quests are in the form of questions and since this is a SG, there isn't any point system to reward correct or incorrect options. As such, players must navigate the city and answer these questions to finish the game and upon answering each question, the players are redirected to scenes to visualize the impact of their choices. In this way, they asses if they have contributed to the city's progress or decline and thus define what is their vision of their ideal city.

To create this digital word, since for each answer the players are presented with the impact of their actions, they grasp the importance of socio-political discussion and the multiple parameters that define a democratic society. Analytically, E-polis is not just another game to collect data from players but can serve as a research tool that helps players understand how society works and visualize the decision-making process, [16].

III. TECHNICAL NOVELTY

Serious games (SGs) are digital games primarily focused on player training and education. Beyond unique graphics, a compelling design architecture and innovative goals are crucial for an SG to capture player interest in today's market.

Our game allows users to make choices (without the option to skip) about their vision of an ideal city. These choices are presented through quests or dilemmas encountered while navigating the city streets. Since this is a serious game and not a traditional game, there are no points, rewards, a publishing system, or a high-score feature. Players are not focused on finding correct or incorrect answers but rather on considering their actions and responses within the presented scenarios.

Our goal was to simulate how a digital democracy might function if citizens could instantly see the consequences of their actions on their city. As mentioned earlier, our game design is heavily influenced by recent literature and the MIT design standard. Core to our beta version and design components is the way we propose dilemmas and set up gameplay, drawing inspiration from these sources: [17], [18], [19], [20], [21], [22], [23], [24]. Our research primarily focused on serious games (SGs) and digital games within the genres of ubiquitous computing and human-computer interaction.

Our digital game novelty is the following:

1. **Technical**: we utilized the Unity game engine. Due to its comprehensive and supportive framework with strict adherence to DRY principles (Don't Repeat Yourself), our technical contribution focuses on software entity abstraction and interaction rather than raw code implementation. In essence, we developed a middleware with well-defined layers based on a software-oriented architecture. This middleware facilitates both game engine features and separates the actual application from the data extraction, transformation, and loading (ETL) technique used:
    a. Our primary goal was to prioritize low-power and resource-efficient data storage for user responses. Initially, we utilized CSV and JSON format datasets. We then transitioned to SQLite, [25], due to its user-friendly support, easy updates, maintainability, and scalability for future game releases. Notably, these data storage solutions were not pre-integrated with the Unity engine editor. We developed them internally to extend the editor's capabilities.
    b. Similar to existing solutions, [26], [27], we focused on the application layer to automate task creation. These tasks include managing player behavior, specifically object instances and their corresponding prefabs within our graphical infrastructure.
    c. We developed rules to categorize player actions based on their choices. These

rules determine the new scenes and levels players encounter to understand the consequences of their decisions. Balancing this with low-power device compatibility for a broader audience was challenging. Our transition mechanism utilizes a combination of pre-defined player behaviors and additional data points. These data points include:

i. Points of interest (POIs) visited by the player
ii. Time spent on difficult dilemmas
iii. Impact of choices on the player (tracked through location and orientation)
iv. By analyzing these data points

d. Scene transition mechanism: For each dilemma (quest), the player is transferred to a new question screen. After answering, the player is redirected to the original screen, at the same coordinates, but the graphics of the original scene change interactively based on the answer. This is imperative to our gameplay, as it helps the player to construct a game world based on their choices. This way, we do not create and destroy scenes, but rather preload the main scene of our game and make temporary transitions to other scenes, which define which game objects will be triggered to generate or destroy a certain area of the town. This mechanism enables us to isolate the player from the current scene and, based on their answer to a question (the response is mandatory to advance the gameplay), generate a new version of the scene with different graphics.

e. Memory Trade-off via Dynamic Building Structure Adjustment: We've developed a novel method for presenting buildings in our game that dynamically adapts the displayed structure based on player choices. This ensures that only the most relevant building representations are loaded into memory, optimizing memory usage without compromising gameplay. Within each city block, players encounter five potential variants of the same building, activated based on their responses. By selectively activating only one variant per block, we significantly reduce memory consumption during gameplay, enabling a more immersive and engaging gaming experience. This optimization strategy balances visual fidelity and memory efficiency.

2. **Theoretical**: we propose a novel generic computational framework for social digital games. This framework takes into account several key properties, including how social relationships might influence player decisions in response to pre-defined questions within a quest system. This serves as an interesting case study for analyzing how players envision and conceptualize their ideal society. The framework's flexibility allows future researchers to adapt the quests to explore different issues related to the concept of an ideal community, as referenced in [28], [29].

IV. GAME ENGINES LITERATURE REVIEW

A. Overview of Digital and Educational Game Development

Since the game's purpose is educational, there was no need for high computer resources or complex graphics and animations. There are many different game engines available. Educational games (similar to our game) are typically lightweight and do not require complex rendering mechanisms. As such, most serious games in the recent literature either use pre-existing solutions and frameworks, simple JavaScript code, and frameworks, or they are hosted on a cloud server.

The key element to consider is the gameplay, specifically whether it requires live, quick, and interactive interaction with the player or a more laid-back approach where the player mainly reads text dialogues and selects answers. The recent trend has been to use JavaScript frameworks or to extract the game to a WebGL version. This is because it is important to have a rapid prototype ready for play as soon as possible, and then to iterate on the game's development process to add new features.

The digital game industry adopts a Rapid Application Development (RAD) approach that focuses on quick feedback from players, rather than a specific release plan for features. Instead, the RAD approach aims to release multiple updates with quality-focused new features throughout the development cycle. This means that a prototype is generated based on user design requirements which is then constantly refined, according to player interaction and the expansion of built modules. This way, game developers can design and construct applications that are agile, flexible, and scalable.

B. Industry Game Engines

As our game is not a “point-and-shoot” (i.e., it requires more actions than simply pointing and clicking on options), we studied the following game engines:

- **Unity**: known for its large community of supporters and ease of use. It is arguably one of the most popular engines in the industry and can develop both 2D and

3D games, [30].

- **Unreal Engine**: known for its large community of supporters and its capabilities to create high-end 3D games, realistic graphics, and physics, [31].
- **Amazon Lumberyard**: known for its ease of use and its -particularly- large library of assets. This is a free game engine that is based on CryEngine, [32].
- **AppGameKit**: easy to use and preferred by beginners and hobbyists. This game engine uses the BASIC (Beginners' All-purpose Symbolic Instruction Code) programming language, [33].
- **Armory3D**: known as a free and open-source game engine that is particularly used by AAA and indie developers. It uses C++ and Rust scripting languages, [34].
- **Babylon.js**: known for its ease of use and performance, this free and open-source game engine is used to create 3D games for the web, [35].
- **Buildbox**: known for its ease of use and its drag-and-drop interface function to create games. Due to this functionality, it is popular with beginners and indie developers, [36].
- **Cocos Creator**: known for its portability and performance, this free and open-source game engine is used to create 2D and 3D games, [37].
- **Cocos2d-x**: popular for mobile game development. This free and open-source game engine uses C++, Lua, and JavaScript, [38].
- **Corona SDK**: known for mobile game development. This game engine uses Lua, [39].
- **CryEngine:** known for its complex capabilities and properties, this engine is selected by many AAA developers, as well as indie developers. This engine uses C++ and Lua, [40].
- **Defold**: ease of use and small sized. This free and open-source game engine is used to create 2D games, [41].
- **Flame Engine**: this engine is used by indie developers. It is free (open-source), and uses C++ and Lua, [42].
- **Gambas**: Similar to AppGameKit, this engine is known for its ease of use and is preferred among beginners and hobbyists. This game engine uses the BASIC (Beginners' All-purpose Symbolic Instruction Code) programming language, [43].
- **GameMaker Studio 2**: Like Buildbox, this engine is known for its ease of use and its drag-and-drop interface function to create games. Due to this functionality, it is popular with beginners and indie developers. It uses a GML scripting language for its drag-and-drop interface, [44].
- **Gdevelop**: this engine is easy to use, thus it is preferred by beginners and indie developers. This is a free and open-source game engine that uses a visual programming language, [45].
- **Godot**: known for its ease, it is used by beginners and indie developers. It is considered one of the most upcoming engines. It was first introduced in 2014 but it has a active community. It is free to use and has an open access license that uses GDScript, a Python-inspired scripting language for development, [46].
- **Leadwerks**: known as a free and open-source game engine that is particularly used by AAA developers, and indie developers. It uses C++, [47].
- **Ogre**: is a well-established game engine that is highly regarded and used in the indie development market that uses C++, thus enabling fast execution times and less resource-intensive computing devices, [48].
- **Open3D Engine**: an open-source and free-to-use and develop/distribute engine that enables developers to create complex games. Similar to Orgre, it is a well-established game engine that is highly regarded and used in indie development that uses C++ and Python, [49].
- **Panda3D**: known for its flexibility and performance. It is a free and open-source game engine that is used to create 3D games, [50].
- **Solar2D**: known as a free and open-source game engine that focuses on portability and ease of use known. It is create mostly used for 2D game development for mobile devices, [51].
- **Stencyl**: known as a free and open-source game engine that focuses on ease of use. It is mostly known for its block-based programming language paradigm that due to its gradual learning curve has made it popular with students and educators worldwide, [52].
- **Stride**: is a well-established game engine that is highly regarded and used in the indie development market that uses known C++ and Lua, [53].
- **Torque 3D**: similar to Stride, it is a well-established game engine that is highly regarded and used in the indie development market It uses C++ and Lua, [54].
- **Torque**: known as a free and open-source game engine that focuses on flexibility and community support during development. It can be used for 2D and 3D game development known, [55].
- **Unigine**: is a well-established game engine that focuses on known realistic graphics and physics. This engine is free to use and its open-source to license is usually used for 3D game development focusing on high-end devices, [56].

C. Industry Game Engines: Literature Review

This section aims to provide a top-to-bottom approach to the game engines used in the industry. Specifically, we present in detail a comparison of the most widely used and recognized game engines presented in the previous chapter. Then, we showcase our in-detail comparison in Table I which focuses on the different multimedia entity modules (effects, graphics sounds, etc.) that are used in each game engine. This way we

highlight the differences between each engine. For example, Unity and Unreal Engine are known for their advanced and complex rendering capabilities, while CryEngine has realistic character animations.

Moreover, we present an analysis of the different technologies used in each game engine (Table II). We provide an overview of each software framework type, including 3D support, physics engine, and target platforms. This information is crucial when deciding which engine to use to develop a game, as it is closely related to the unique properties of the project as well as time and budget constraints. It is worth noting that most game engines are open source, except for Corona SDK, Torque, AppGameKit, GameMaker Studio 2, Buildbox, Stride, Amazon Lumberyard, Unity, Unreal Engine, CryEngine, and Leadwerks, which use proprietary software licenses.

TABLE I. GAME ENGINES COMPARISON BASED ON RENDERING, ANIMATION, AND SOUND PROPERTIES

Game Engine	Rendering	Animation	Sound
Babylon.js	WebGL-based rendering Support for a variety of lighting and shading models	Skeletal animation system	Support for Web Audio API and other audio libraries
Gdevelop	Sprite-based rendering Support for a variety of lighting and shading models		
GameMaker Studio 2			
Defold			
Cocos2D-x			
Corona SDK	Scene graph-based rendering Support for a variety of lighting and shading models	-	Support for OpenAL and other audio libraries
Unreal Engine	Physically based rendering, real-time lighting, shadows, and reflections Support for a wide range of rendering pipelines	Detailed animation system with support for skeletal, facial, and procedural animation	Built-in support for spatial audio, sound effects, and music
Unity			
Leadwerks			
Godot			
Flame Engine			
CryEngine			
Cocos Creator			
Armory3D			
Amazon Lumberyard			
Gambas	Basic rendering capabilities	Basic animation capabilities	Basic sound capabilities
Buildbox			
AppGameKit			

TABLE II. GAME ENGINES COMPARISON BASED ON SOFTWARE PROPERTIES (3D SUPPORT, PHYSICS, PLATFORM)

Game Engine	3D support	Physics engine	Build Platform
Armory3D	Yes	Bullet	Windows, macOS, Linux, Android, iOS, Web, WebGL
Corona SDK	Yes	Chipmunk	iOS, Android
Babylon.js	Yes	Cannon	Web, WebVR
Gambas	Yes	N/A	Windows, Linux
Panda3D	Yes	Bullet	Windows, macOS, Linux
Torque 3D	Yes	Bullet	
Torque	Yes	Bullet	
Solar2D	No	N/A	
AppGameKit	No	Newton	
Unigine	Yes	PhysX	
Ogre	Yes	Bullet	
Flame Engine	Yes	Bullet	Windows, macOS, Linux, Android, iOS
Open3D Engine	Yes	Bullet	Windows, macOS, Linux, Android, iOS, Web, WebGL
Gdevelop	Yes	Bullet	
Stencyl	Yes	Bullet	
GameMaker Studio 2	Yes	Bullet	
Godot	Yes	Bullet	
Cocos2D-x	Yes	Bullet	
Defold	Yes	Bullet	
Buildbox	No	N/A	
Cocos Creator	Yes	PhysX	
Stride	Yes	PhysX	
Amazon Lumberyard	Yes	PhysX	Windows, macOS, Linux, Android, iOS, Web, WebGL, Xbox One, PlayStation 4
Unity	Yes	PhysX	Windows, macOS, Linux, Android, iOS, Web, WebGL, Xbox One, PlayStation 4, Nintendo Switch, Stadia
Unreal Engine	Yes	PhysX	Windows, macOS, Linux, Android, iOS, Web, WebGL, Xbox One, PlayStation 4, PlayStation 5, Xbox Series X/S, Nintendo Switch

V. IMPLEMENTATION

In the following sections, we will present the software architecture of our implementation. First, we explain the scene transition mechanism we developed to store the graphics, physics, and properties of our game. Subsequently, based on Unity's architectural design principles, we showcase the scenes of our game in detail. Finally, we focus on the data acquisition

element of our game and the different things we need to consider when implementing the extract, transform, and loading solutions.

A. Graphics Rendering and Scene Transition Mechanism in Unity

1) Overview of the Mechanism

As mentioned in the previous sections, the technical novelty of this article is the scene transition mechanism. This functionality allows players to transition to different scenes, interact with quests via dialogue, and shape the town's buildings. This is of the utmost importance, as it defines how the players' gameplay constructs the unique city and its characteristics, which can later be used to study their answers. It is worth noting that the purpose of the game is not only for players to complete a set of tasks, but also to study their behavior and choices when presented with a visual, real-world outcome of their options.

Firstly, we needed to determine the overall components needed to construct this mechanism, which was to compare two different architectural choices. The first approach involved cloning the scene. This meant that each time the player interacted with a point-and-click quest or went through dialogue, we cloned the current scene's graphics and transitioned them to the next scene. Afterward, based on their answer, we would reload the original scene and compare the objects from the hierarchy to recreate the game world as it was before, and then add the changes based on each question. Later, we decided that it would be far more performance-effective to store the body position (X, Y, Z axis) and the head rotation (Euler angles) to restore the player to the same position and scene to witness how their choice changed or redefined it.

We also tried to enhance the performance of our solution by shifting the canvas statuses. Specifically, we disabled the canvas and saved the game object of the player (mesh capsule) coordinates. This happened via a prefab object that automatically generated the necessary scripts to store the player's coordinates when the player entered its point of view. The prefab object also disabled the current canvas and enabled a new one. Upon collision with this prefab, a switch is generated which is assigned to a specific quest (thus, no quest can be presented more than once). The player is presented with a task and must complete it to proceed. Upon successful completion, we store their answer and destroy the prefab, thus disabling the quest from our overall gameplay. The same mechanism was later via actual graphics, instead of a prefab of a canvas.

2) Algorithms of the Scene Transition Mechanism

The inner work of this algorithm's first implementation is presented in Table III. Similarly, in Table IV we present the extended and improved version where instead of prefabs, we rendered actual building blocks based on the quest's completion. This way, the player constructed the future based on his answer and shaped the city's buildings. Lastly, it is noted that the algorithm presented in this article uses Pseudocode, for explanation purposes only. The actual code was written in Unity, thus as per the game engine's requirements it is in C# scripts.

TABLE III. INITIAL ALGORITHM USED FOR DILEMA WITH PREFAB COLLISION

```
# Disable all canvases
disable_all_canvases()

# When entering the point of view of a prefab:
# Store the player's coordinates and Euler angles
store_player_coordinates_and_euler_angles()

# Create a boolean switch to enable the dilemma
enable_dilemma_switch()

# While the player is in the point of view of the
prefab:
while in_prefab_point_of_view():

    # Check if there is a collision
    if collision_occurs():

        # Enable the dilemma switch
        enable_dilemma_switch()

    # If the dilemma switch is enabled:

    if dilemma_switch_enabled():

        # Get the player's answer to the dilemma
        get_player_answer()

        # Disable the dilemma switch
        disable_dilemma_switch()

# When leaving the point of view of the prefab:

# Destroy the prefab
  destroy_prefab()
```

Regarding Table III, we specify the modules of each function as follows:

- *disable_all_canvases()* function disables all canvases in the game.
- *store_player_coordinates_and_euler_angles()* function stores the player's coordinates and Euler angles in variables. The Euler angles represent the player's head orientation and body position.
- *enable_dilemma_switch()* function sets the dilemma_switch variable to True.
- *in_prefab_point_of_view()* function returns True if the player is currently in the point of view of a prefab, and False otherwise.
- *collision_occurs()* function returns True if the player collides with something, and False otherwise.
- *disable_dilemma_switch()* function sets the dilemma_switch variable to False.
- *get_player_answer()* function prompts the player to answer the dilemma and returns their answer.
- *destroy_prefab()* function destroys the prefab that the player is currently viewing.

TABLE IV. ADVANCED ALGORITHM USED FOR DILEMMA WITH GRAPHIC RENDERING

```
# Disable all canvases
disable_all_canvases()

# When entering the point of view of a prefab:
# Store the player's coordinates and Euler angles
store_player_coordinates_and_euler_angles()

# Create a boolean switch to enable the dilemma
enable_dilemma_switch()

# While the player is in the point of view of the
prefab:
while in_prefab_point_of_view():

    # Check if there is a collision
    if collision_occurs():

        # Enable the dilemma switch
        enable_dilemma_switch()

    # If the dilemma switch is enabled:

    if dilemma_switch_enabled():

        # Get the player's answer to the dilemma
        get_player_answer()

        # Disable the dilemma switch
        disable_dilemma_switch()

# When leaving the point of view of the prefab:

# Destroy the prefab
  destroy_prefab()
```

Regarding Table IV, we specify the modules of each function as follows:

- *load_scene_view_from_above()* - This function loads a scene that shows the new city that the player has created from above.
- *check_for_modules_of_completion()* - This function checks if the player has completed all of the required quests and tasks. This is important because it ensures that the player has fully experienced the game moving through all its phases.
- *destroy_previous_objects_and_scenes_for_performance()* - This function destroys any objects and scenes that are no longer needed. This frees up memory and improves the game's performance.
- *check_database_answers_and_connectivity()* - This function checks if the players' answers to the quests have been saved correctly and that they are still connected to the game server. This is important because it ensures that the players' progress is preserved and that they can play the game without interruptions.
- *check_screenshot_functionality()* - This function checks if the player can take screenshots of the game. This is important because it allows the player to capture and share their gaming experience.
- *save_game()* - This function saves the player's progress. This ensures that the player can pick up where they stopped if needed.

B. Game Engine Used

After studying the most widely used game engines in the industry (OpenSimulator, Godot, CryEngine, GameMaker), and taking into account Table I and Table II, we concluded that the most interesting game engines based on the specific elements of our game (i.e. a light, not resource intensive interaction educational game) were Unity and Unreal engine.

Several game engines exist, each with its strengths and weaknesses. After evaluating both Unity and Unreal Engine, we ultimately chose Unity for this project. While Unreal Engine might boast faster execution times, Unity's developer-friendliness and platform independence were decisive factors. Unity's versatile export settings allow developers to target various platforms and operating systems more easily compared to Unreal Engine, [57]. For the development of the game described in this paper, we utilized Unity version 2020.3.24f1 with a private license. The hierarchy of our game objects is the following (Figure 1):

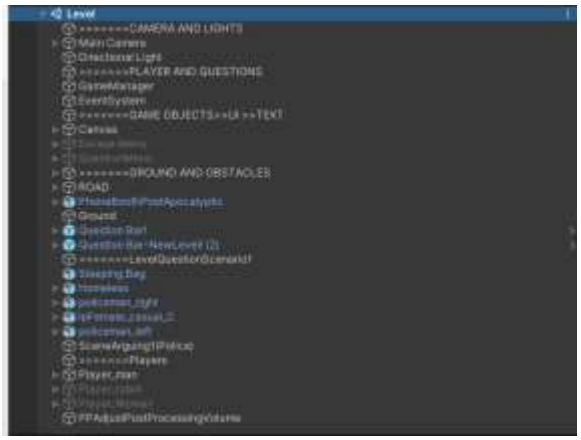

Fig. 1 Our SG's Unity's hierarchy view as presented in the Unity Editor, [58]

Player data will be collected and analyzed using IBM SPSS Statistics software. We anticipate a low data volume, with a single test group playthrough generating less than 1,000 records due to our focus on efficient data storage techniques. To extract data, we incorporated a mesh grip terrain under the graphics of each scene, allowing the player to move to specific coordinates. This enables us to record any movements in any direction. A view of the developers' framework is presented in Figure 2. It showcases the game's axis, the terrain divided into a grid, and the actual buildings used.

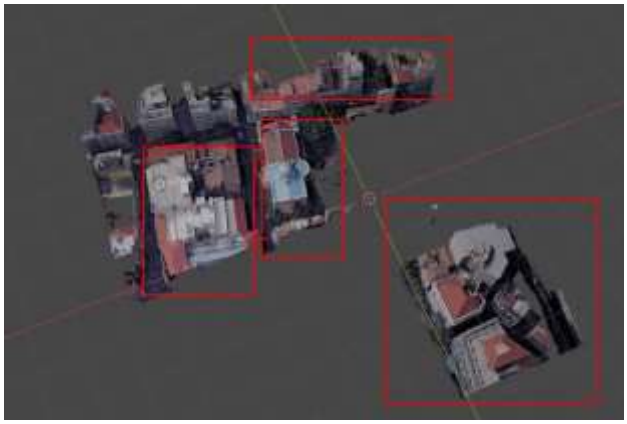

Fig. 2 The Unity Editor gird to the developer's view regarding the movement of the player for saving their gameplay actions, [58]

Our game generates two main datasets in the form of tables. The first table, titled "Player Actions," stores information about player choices. This includes the avatar's name, the number of the quest encountered, a short description of the quest text, the player's answer to the quest, and a timestamp for when the answer was submitted. The second table, focused on "Player Movements," tracks data on player movement throughout the game. It captures the X, Y, and Z coordinates of the player's location on the map, the orientation of the player's head using Euler angles for each axis, and a timestamp for each record collected at a preset sampling rate.

This data can be exported in various formats, including CSV, XML, YAML, or JSON, allowing for interaction with most industry-standard data analysis software tools.

VI. Game development middleware

A. Middleware for Game Deveopers

Middleware is a software layer between the game engine and the underlying operating system and hardware, [59]. It provides a set of services that game developers can use to develop games without worrying about how the game engine works, [60]. In this final section, we present the main architectural design principles of the middleware we developed to construct a serious game, using the Unity game engine. The mechanism we present is the main contribution of our middleware architecture; instead of using the asset-prefab-scene and script segregation, we expanded our middleware as follows:

- **Platform layer**: This layer provides a common interface to the underlying operating system and hardware. This allows game developers to write code that is portable to different platforms. This mainly incorporates the separation of concerns to the asset-prefab-scene and script, as mentioned in the following review paper, [29].
- **Engine layer**: This layer provides a set of services that are common to most game engines, such as scene management, rendering, and physics, as well as modules. This includes Section 5.1 Scene Transition Mechanism and the algorithms we developed.
- **Game layer**: This layer provides services that are specific to the game, such as character control, AI, and gameplay mechanics. This includes the analysis of the Acquiring Player's Data module.
- **Application layer**: This layer provides services that are specific to the application, such as user interface, input handling, and network communications. While we used Unity, properties are available in previous sections.

B. Generic Content Middleware for Game Development

Similarly, middleware can also be defined based on the operations that were described in this article. Since this is a software implementation using the Unity game engine, we briefly present the front-end and back-end layers. These layers can be incorporated into the above. However, to promote the functionality of the project and because this is an educational serious game, we chose to expand them into separate layers:

1. The **front-end layer** is responsible for all user interactions, such as *rendering* the game world and handling user input.
2. The **back-end layer** is responsible for all the *logic* of our game, such as scene transitions, different game states for animators, and navigation through the game world to complete quests.

Specifically, while developing our generic game development middleware, we have focused on separating our entities based on their application processes. In this way, we segregated our initial monolithic application into many separate independent layers thus making our system modular and easier to scale and maintain.

1) Front-end layer

- *Prefab design tools*: the prefabs are needed to reduce the use of the same objects multiple times in the hierarchy. As such, the case of a tool specific for designing prefabs with a visual editor for scripting and debugging is needed.
- *Terrain meshing and structure monitoring*: this layer serves as an intermediate for the necessary services that generate the terrain (meshes) of our game.
- *Coordinate and action monitoring*: Provides the processes to track and monitor the location (coordinates) of the players' actions. This also includes NPCS and any other object that may interact with the player.
- *Graphics and scene management*: Provides processes for rendering the game graphics. It is also responsible for handling the scene transition mechanism, the lightning mechanism, and the re-use of prefabs (copies of new objects and destroying the unused ones).

- *AI movement and camera interaction*: Provides the processes to modify the movement of AI playable characters and the re-adjustment of the player's main camera component.
- *Google Maps 3D model extraction*: Provides displaying for loading and extracting 3D models based on Google Maps location and importing them as fbx to the game project.
- *UI canvas for questions and dialogue*: Provides processes for developing and displaying UI textbox thus simulating quests and a dialogue based for.
- *Tree and building rendering*: Provides processes to render the game world in general and the dynamic décor of our world (rendering of trees buildings, etc.).
- *Particle effects and sound effects*: Provides processes creating and handle the particle effects of our prefabs when answering a dilemma/quest and the respective sound effects when creating and destroying each prefab object.

2) *Back-end layer*

- *Prefabs*: Provides for the processes to manage prefabs from generating/destroying them to defining their properties.
- *Points of view*: Provides the processes for managing the player camera. This means defining the point of view of an individual, or an AI NPC and its range of actions.
- *Zone triggering*: Provides processes to handle the range of the triggering of game events. This must be achieved via defining zones of action where the prefabs will operate.
- *Switch actions*: Provides processes for creating automation switch operations. This means that when the player enter a specific zone, certain operations will be triggered that will act as switches for each quest. This means that for a new operation to load, an old one will need to be deactivated thus acting as a true-false switch for the layering and order of each operation.
- *Quest and dilemma agents*: Provides processes responsible for handling different quests. This means that the dilemmas presented to each player must initiate and end with a preset game object that will be closely linked with an action and event.
- *NPC action management*: Provides processes to handle the overall actions of NPCS as well as the permeated interactions and action id that can be instantiated.
- *Player interaction scenarios*: Provides all the necessary processes for the actual player(s) interactions for completing the game. This means providing a logic/scenario on how to start/end the game and how a player will understand the game logic, rules, and physics applied.
- *Open vs Close answered questions*: The prefab mechanism allows us to designate questions as either one-time or re-answerable, enabling them to appear multiple times throughout the game. These responses are critical for our analysis of player behavior and the evolving structure of the buildings. This flexibility is paramount for both the statistical analysis that may be incorporated at the end of the game (i.e., augmenting sample size) and the dynamic definition of the game's blueprint. By enabling or disabling re-answerable questions, we can adjust the range of choices and properties for the generated buildings, influencing the overall structure of each gameplay experience while also optimizing the database and computer resources required for smooth operation.

VII. Conclusion

This paper presents our novel serious game (SG) and the middleware we developed to create it. We also explore a generic middleware framework for separating front-end and back-end processes for objects and mechanics during development within the Unity game engine. As for the SG's use for social research, it offers scientists the ability to showcase the impact of player actions. This allows social researchers to examine the contrasting effects of right and left-wing choices within the game's simulated environment. While recent literature often focuses on the outcomes and answer quality, we aim to bridge the gap by explaining the development tools and methods behind our game.

We hope future developers will consider implementing our proposed middleware layer(s). This approach could provide a more universally accepted way to develop games using game engines. The middleware optimizes and automates the creation of communication and application/game layers for each SG, fostering efficiency, [61], [62], [63], [64].

Our next steps involve recruiting more players to gather data and feedback on the game's graphics and quest design. This will help us refine the experience for future iterations. To aid future social science research, we plan to develop a basic clustering algorithm using K-Means or DBSCAN on player data for each task. This will allow researchers to identify patterns in player behavior and decision-making. We also propose creating an online tool that exports the final city structure based on player choices. This will allow players to compare the impact of their decisions and those of others on the city's development. For broader accessibility, we'll explore exporting the game to WebGL or another cloud-based architecture, enabling a wider global audience to participate. Finally, to enhance user engagement, we're investigating transitioning the game from keyboard controls to VR headsets. This would involve adapting the game for a virtual 3D world.

These future steps hold promise for enriching the player experience, advancing social science research using games, and fostering a more globally inclusive platform.

ACKNOWLEDGMENT

. This article is an extended version of the conference paper titled "E-polis: A Serious Game for the Gamification of Sociological Surveys," published in the proceedings of the International Conference on Applied Mathematics & Computer Science, [58].

References

[1] Chowdhury, J., Sarkar, A., Mahapatra, K., & Das, J. K. (2024). More-than-Moore Steep Slope Devices for Higher Frequency Switching Applications: A Designer's Perspective. *Physica Scripta*. https://www.doi.org/10.1088/1402-4896/ad2da2

[2] Gazis, A., & Katsiri, E. (2021). Smart home IoT sensors: Principles and applications a review of low-cost and low-power solutions, *International Journal on Engineering Technologies and Informatics*, 2(1), 19-23. https://doi.org/10.51626/ijeti.2021.02.00007

[3] Mudaliar, M. D., & Sivakumar, N. (2020). IoT based real-time energy monitoring system using Raspberry Pi. *Internet of Things*, 12, 100292. https://doi.org/10.1016/j.iot.2020.100292

[4] Campagnaro, F., Steinmetz, F., & Renner, B. C. (2023). Survey on low-cost underwater sensor networks: from niche applications to everyday use. *Journal of Marine Science and Engineering*, 11(1), 125. https://doi.org/10.3390/jmse11010125

[5] Sriraam, N., Srinivasulu, A., & Prakash, V. S. (2023). A Low-Cost, low-power flexible single-lead ECG textile sensor for continuous monitoring of Cardiac Signals. *IEEE Sensors Journal*. https://doi.org/10.1016/j.icte.2024.03.003

[6] Ozkan-Ozay, M., Akin, E., Aslan, Ö., Kosunalp, S., Iliev, T., Stoyanov, I., & Beloev, I. (2024). A Comprehensive Survey: Evaluating the Efficiency of Artificial Intelligence and Machine Learning Techniques on Cyber Security Solutions. *IEEE Access*. https://doi.org/10.1109/ACCESS.2024.3355547

[7] Rogers, K., Karaosmanoglu, S., Altmeyer, M., Suarez, A., & Nacke, L. E. (2022, April). Much realistic, such wow! a systematic literature review of realism in digital games. *CHI Conference on Human Factors in Computing Systems* (pp. 1-21), New Orleans, LA, USA. https://doi.org/10.1145/3491102.3501875

[8] Tecedor, M. (2024). Digital storytelling: changing learners' attitudes and self-efficacy beliefs. *Applied Linguistics*, 45(1), 65-87. https://doi.org/10.1093/applin/amad002

[9] Ishak, S. A., Hasran, U. A., & Din, R. (2023). Media Education through Digital Games: A Review on Design and Factors Influencing Learning Performance. *Education Sciences*, 13(2), 102. https://doi.org/10.3390/educsci13020102

[10] Chowdhury, M., Dixon, L. Q., Kuo, L. J., Donaldson, J. P., Eslami, Z., Viruru, R., & Luo, W. (2024). Digital game-based language learning for vocabulary development. Computers and Education Open, 6, 100160. https://doi.org/10.1016/j.caeo.2024.100160

[11] Peterson, M. (2023). Digital simulation games in CALL: A research review. *Computer Assisted Language Learning*, 36(5-6), 943-967. https://doi.org/10.1080/09588221.2021.1954954

[12] Susanti, A., Darmansyah, A., Naqsyahbandi, F., & Muktadir, A. (2024). Analyzing student learning style profiles for differentiated learning in merdeka curriculum in elementary schools. Cendikia: Media Jurnal Ilmiah Pendidikan, 14(3), 209-223. https://doi.org/10.35335/cendikia.v14i3.4589

[13] Arjoranta, J. (2019). How to define games and why we need to, *The Computer Games Journal*, 8(3-4), 109-120. https://doi.org/10.1007/s40869-019-00080-6

[14] Gumbi, N. M., Sibaya, D., & Chibisa, A. (2024). Exploring Pre-Service Teachers' Perspectives on the Integration of Digital Game-Based Learning for Sustainable STEM Education. *Sustainability*, 16(3), 1314. https://doi.org/10.3390/su16031314

[15] Tang, Z., & Kirman, B. (2024). Exploring Curiosity in Games: A Framework and Questionnaire Study of Player Perspectives. International Journal of Human–Computer Interaction, 1-16. https://doi.org/10.1080/10447318.2024.2325171

[16] Tekinbas, K. S., & Zimmerman, E. (2003). Rules of play: Game design fundamentals, *MIT Press*. https://mitpress.mit.edu/9780262240451/rules-of-play/ [Accessed on 08/04/2024]

[17] Seaborn, K., & Fels, D. I. (2015). Gamification in theory and action: A survey, *International Journal of human-computer Studies*, 74, 14-31. https://doi.org/10.1016/j.ijhcs.2014.09.006

[18] Taylor, T. L. (2018). Watch me play. In Watch Me Play, *Princeton University Press*. https://doi.org/10.1515/9780691184975

[19] Tsai, Y. L., & Tsai, C. C. (2020). A meta- analysis of research on digital game-based science learning, *Journal of Computer Assisted Learning*, 36(3), 280-294. https://doi.org/10.1111/jcal.12430

[20] Gupta, A., Lee, S., Mott, B., Chakraburty, S., Glazewski, K., Ottenbreit-Leftwich, A., & Lester, J. (2024, March). Supporting Upper Elementary Students in Learning AI Concepts with Story-Driven Game-Based Learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 38, No. 21, pp. 23092-23100), Vancouver, Canada. https://doi.org/10.1609/aaai.v38i21.30354

[21] Breien, F. S., & Wasson, B. (2021). Narrative categorization in digital game-based learning: Engagement, motivation & learning, *British Journal of Educational Technology*, 52(1), 91-111. https://doi.org/10.1111/bjet.13004

[22] Schöbel, S., Saqr, M., & Janson, A. (2021). Two decades of game concepts in digital learning environments–A bibliometric study and research agenda, *Computers & Education*, 173, 104296. https://doi.org/10.1016/j.compedu.2021.104296

[23] Gui, Y., Cai, Z., Yang, Y., Kong, L., Fan, X., & Tai, R. H. (2023). Effectiveness of digital educational game and game design in STEM learning: a meta-analytic review, *International Journal of STEM Education*, 10(1), 1-25.https://doi.org/10.1186/s40594-023-00424-9

[24] Negahban, A. (2024). Simulation in engineering education: The transition from physical experimentation to digital immersive simulated environments. Simulation, 00375497241229757. https://doi.org/10.1177/00375497241229757

[25] Gaffney, K. P., Prammer, M., Brasfield, L., Hipp, D. R., Kennedy, D., & Patel, J. M. (2022). Sqlite: past, present, and future, *Proceedings of the VLDB Endowment*, 15(12), 3535-3547, Vancouver, Canada . https://doi.org/10.14778/3554821.3554842

[26] Linowes, J. (2015). Unity virtual reality projects, *Packt Publishing Ltd*. https://www.packtpub.com/product/unity-virtual-reality-projects-second-edition/9781788478809 [Accessed on 08/04/2024]

[27] Nusrat, F., Hassan, F., Zhong, H., & Wang, X. (2021, May). How developers optimize virtual reality applications: A study of optimization commits in open source unity projects, *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)* (pp. 473-485). IEEE, Madrid, Spain. https://doi.org/10.1109/ICSE43902.2021.00052

[28] E-polis http://www.e-polis.pspa.uoa.gr. [Accessed on 21/03/2024]

[29] Gazis, A., & Katsiri, E. (2023). Serious Games in Digital Gaming: A Comprehensive Review of Applications, Game Engines and Advancements, *WSEAS Transactions on Computer Research*, 11, 10-22. https://dx.doi.org/10.37394/232018.2023.11.2

[30] Unity Technologies Github Repository. https://github.com/Unity-Technologies. [Accessed on 21/03/2024]

[31] Unreal Engine Github Repository. https://github.com/topics/unreal-engine-4. [Accessed on 08/04/2024]

[32] Amazon Web Services (AWS) Lumberyard Github Repository. https://github.com/aws/lumberyard. [Accessed on 21/03/2024]

[33] TheGameCreators AGK repository. https://github.com/TheGameCreators/AGKRepo. [Accessed on 04/08/2024]

[34] Armory3D GitHub Repository. https://github.com/armory3d/armory. [Accessed on 21/03/2024]

[35] Babylon.js GitHub Repository. https://github.com/BabylonJS/Babylon.js/. [Accessed on 21/03/2024]

[36] Packt Publishing Buildbox-2x-Game-Development GitHub Repository. https://github.com/PacktPublishing/Buildbox-2x-Game-Development. [Accessed on 21/03/2024]

[37] Cocos Engine GitHub Repository. https://github.com/cocos/cocos-engine. [Accessed on 21/03/2024]

[38] Cocos2d-x GitHub Repository. https://github.com/cocos2d/cocos2d-x. [Accessed on 21/03/2024]

[39] Corona Labs Corona GitHub Repository. https://github.com/coronalabs/corona. [Accessed on 21/03/2024]

[40] CryEngine GitHub Repository. https://github.com/ValtoGameEngines/CryEngine. [Accessed on 21/03/2024]

[41] Defold GitHub Repository. https://github.com/defold/defold. [Accessed on 21/03/2024]

[42] Flame Engine GitHub Repository. https://github.com/flame-engine/flame. [Accessed on 21/03/2024]

[43] Gambas GitHub Repository. https://github.com/landv/gambas. [Accessed on 21/03/2024]

[44] GameMaker Studio 2 GitHub Repository. https://github.com/topics/gamemaker-studio-2. [Accessed on 21/03/2024]

[45] GDevelop GitHub Repository. https://github.com/4ian/GDevelop. [Accessed on 21/03/2024]

[46] Godot Engine GitHub Repository. https://github.com/godotengine. [Accessed on 21/03/2024]

[47] Leadwerks GitHub Repository. https://github.com/Leadwerks. [Accessed on 21/03/2024]

[48] OGRE GitHub Repository. https://github.com/OGRECave/ogre. [Accessed on 21/03/2024]

[49] Open3D GitHub Repository. https://github.com/isl-org/Open3D. [Accessed on 21/03/2024]

[50] Panda3D GitHub Repository. https://github.com/panda3d/panda3d. [Accessed on 21/03/2024]

[51] Solar2D GitHub Repository. https://github.com/solar2d. [Accessed on 21/03/2024]

[52] Stencyl GitHub Repository. https://github.com/Stencyl/stencyl-engine. [Accessed on 21/03/2024]

[53] Stride3D GitHub Repository. https://github.com/stride3d/stride. [Accessed on 21/03/2024]

[54] Torque3D GitHub Repository. https://github.com/GarageGames/Torque3D. [Accessed on 21/03/2024]

[55] TorqueGameEngines GitHub Repository. https://github.com/TorqueGameEngines. [Accessed on 21/03/2024]

[56] Unigine GitHub Topic. https://github.com/topics/unigine. [Accessed on 21/03/2024]

[57] Marin-Vega, H., Alor-Hernández, G., Zatarain-Cabada, R., Barron-Estrada, M. L., & García-Alcaraz, J. L.

(2020), A brief review of game engines for educational and serious games development. *Language Learning and Literacy: Breakthroughs in Research and Practice*, 447-469. https://doi.org/10.4018/978-1-5225-9618-9.ch024

[58] Gazis, A., & Katsiri, E. (2023, August). E-polis: A serious game for the gamification of sociological surveys. In 2023 International Conference on Applied Mathematics & Computer Science (ICAMCS) (pp. 154-161). IEEE, Lefkada, Greece. https://doi.org/10.1109/ICAMCS59110.2023.00032

[59] Gazis, A., & Katsiri, E. (2022). Middleware 101, *Communications of the ACM*, 65(9), 38-42. https://dl.acm.org/doi/10.1145/3546958

[60] Gazis, A., & Katsiri, E. (2022). Middleware 101: What to know now and for the future, *Communications of the ACM*, 20(1), 10-23. https://dl.acm.org/doi/10.1145/3526211

[61] Shute, V., & Ventura, M. (2013). Stealth assessment: Measuring and supporting learning in video games (p. 102), *MIT Press*. http://library.oapen.org/handle/20.500.12657/26058 [Accessed on 08/04/2024]

[62] Fernández-Sánchez, M. R., González-Fernández, A., & Acevedo-Borrega, J. (2023). Conceptual Approach to the Pedagogy of Serious Games, Information, 14(2), 132. https://doi.org/10.1016/j.csi.2016.09.014

[63] Damaševičius, R., Maskeliūnas, R., & Blažauskas, T. (2023). Serious games and gamification in healthcare: a meta-review. *Information*, 14(2), 105. https://doi.org/10.3390/info14020105

[64] Pueyo-Ros, J., Comas, J., Säumel, I., Castellar, J. A., Popartan, L. A., Acuña, V., & Corominas, L. (2023). Design of a serious game for participatory planning of nature-based solutions: The experience of the Edible City Game. *Nature-Based Solutions*, 3, 100059. https://doi.org/10.1016/j.nbsj.2023.100059

Contribution of individual authors to the creation of a scientific article (ghostwriting policy)

- Mr Alexandros Gazis, was responsible for the conceptualization, investigation, methodology, software, validation, visualization, writing the original draft, review-editing resources, carrying out simulations, and writing the original draft of this paper.
- Professor Eleftheria Katsiri, contributed to the conceptualization, formal analysis, funding acquisition, investigation, methodology, project administration, resources, supervision, validation, visualization, review, and editing of the original draft.

We confirm that all Authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of funding for research presented in a scientific article or scientific article itself

This work was funded in part by the research project "e-polis of the future", supported by the Hellenic Foundation of Research and Innovation (H.F.R.I.) in the context of the "1st Call for H.F.R.I. (http://www.elidek.gr) Research Projects to Support Faculty Members & Researchers and Procure High-Value Research Equipment" (Project Number: 2617). The authors would like to thank Mr. Gerasimos Kouzelis for providing the research outline of this project and Mr. Orestis Didi for his overall assistance and expertise on the topic.

Conflicts of Interest

The authors have no conflicts of interest to declare that are relevant to the content of this article.