FRI-1.407.1-SSS-I-01

ADVANTAGES AND DISADVANTAGES OF DEVELOPING A SYSTEM FOR INTERACTION BETWEEN THE PLAYER AND THE GAME ENVIRONMENT IN THE GAME ETERNAL VIGIL [17]

Serkan Sadulov, Student
Faculty of Natural Sciences and Education,
University of Ruse "Angel Kanchev"
Phone: +359 87 892 5635
E-mail: Ssadulov@uni-ruse.com

Mustafa Mustafov, Student
Faculty of Natural Sciences and Education,
University of Ruse "Angel Kanchev"
Phone: +359 89 056 9310
E-mail: s216259@stud.uni-ruse.bg

Assoc. Prof. Kamelia Shoilekova, PhD – Supervisor
Department of Informatics and Information Technologies,
University of Ruse "Angel Kanchev"
Phone: +359 88 785 9224
E-mail: kshoylekova@uni-ruse.bg

Abstract: *This paper provides an in-depth analysis of innovative battle system design strategies in video game development, focusing on the use of advanced algorithms and industry best practices. Research focuses on creating dynamic and engaging player-versus-environment encounters by exploring the implementation of various algorithms, such as finite state machines and behavior trees, to design responsive and intelligent behavior of non-player characters (NPCs). The study carefully examined the theoretical basis and practical application of these algorithms in the development of combat mechanisms with the aim of improving player immersion and difficulty. Using modern programming techniques and computational models, this paper presents a holistic approach to building combat systems that facilitate complex interactions and facilitate immersive gaming experiences.*

Keywords: *Video game development, Combat system, Algorithms, Player versus environment, Dynamic interactions, Colliders.*

INTRODUCTION

Developing a system for interaction between the player and the game environment is a key aspect of creating an exciting and engaging gaming experience. In the context of the game "Eternal Vigil," this system plays a crucial role in determining how players interact with the world around them, solve puzzles, overcome obstacles, and progress in the game. This report examines the advantages and disadvantages of developing such a system by analyzing how it affects the gaming experience, technical requirements, and creative possibilities for developers. Understanding these aspects is essential for achieving a balanced and satisfying gaming experience that keeps players' attention and contributes to the game's success in the market.

[17] This publication reflects research from the scientific project 23-FPNO-02 "Investigation of effective knowledge management mechanisms applied in software engineering when creating projects with Agile methodologies" - of the "Scientific Research" fund of Ruse University "Angel Kanchev", 2023.

EXPOSITION

Nature of the Game Environment

The game environment represents the virtual world in which players interact, perform tasks, and experience the story of a given game. It includes various elements such as graphics, sounds, physics, rules, and game mechanics. The game environment is the core of the gaming experience as it defines the context in which players act and react (Fig. 1) [2], [3], [4], [5], [6].

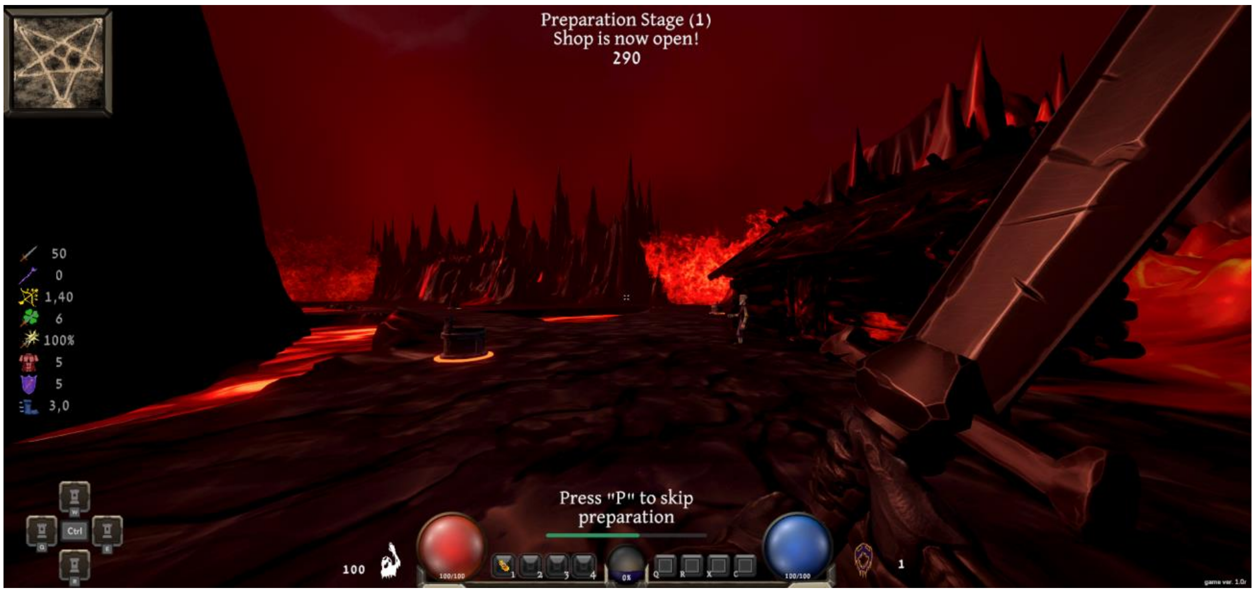
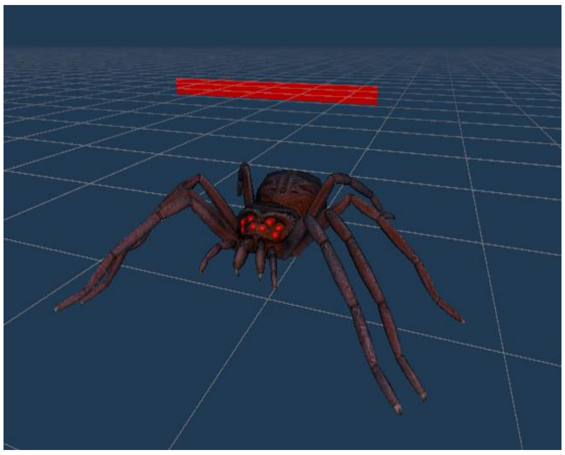


Fig. 1. Game Environment

Main Components of the Game Environment:

1. **Graphics and Visual Elements:**
   a. Textures and Models: These components create the visual appearance of objects, characters, and locations in the game (Fig. 2).
   b. Lighting Effects: They add realism and atmosphere by influencing the perception of objects and spaces.
   c. Animations: The movements of characters and objects that bring the game world to life (Fig. 3).



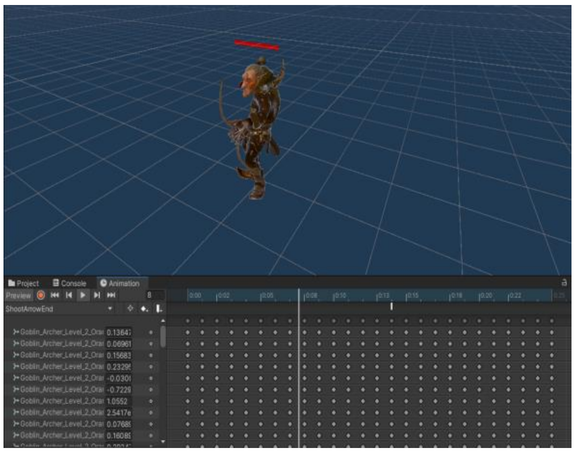


Fig. 2. Spider Model

Fig. 3 Animation Execution

2. **Sound Elements:**
   a. Background Sounds and Music: They complement the atmosphere and enhance the emotional impact of the game environment.

   b. Effects: Sounds that are triggered by specific actions or events such as footsteps, gunshots, or conversations.

3. **Physics and Simulation:**
   a. Laws of Physics: The rules that determine how objects move and interact with each other in the game world.
   b. Collisions: Define how objects collide and react to contact (Fig. 4), with part of the program implementation shown in (Fig. 5).
4. **Interactive Elements:**
   a. Interactive Objects: Elements of the environment with which players can interact, such as doors, switches, collectable items, and NPCs (non-player characters).
   b. Scripts and Events: Programmed actions and reactions that are triggered under certain conditions or events [8], [9].

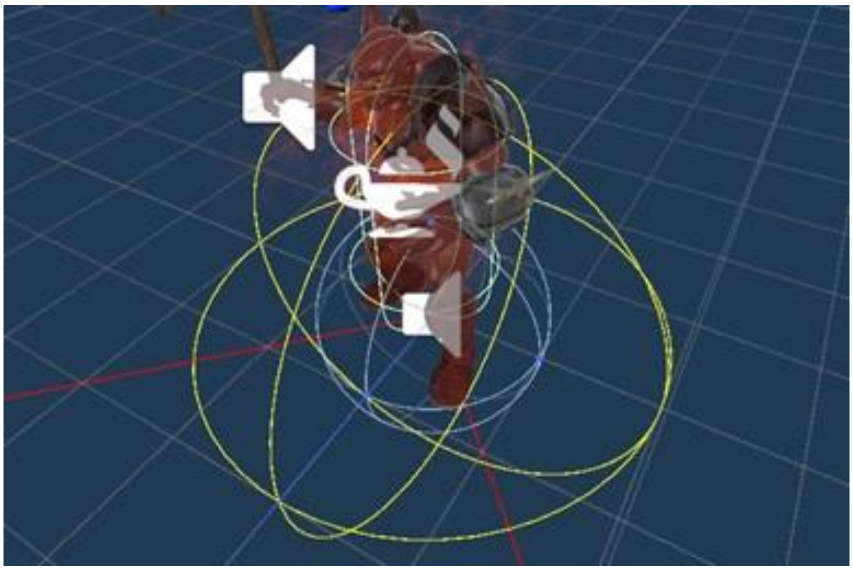
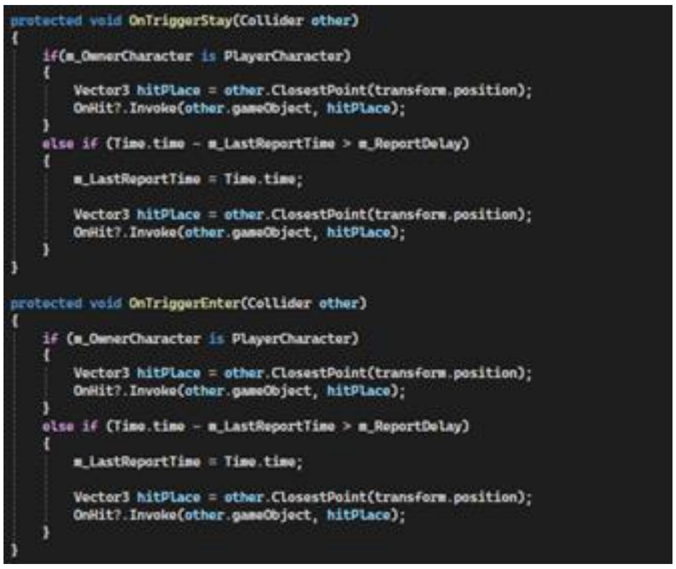


Fig. 4. Colliders of a Game Object

Principle of Operation of the Combat System in "Eternal Vigil":

The combat system in the game is built on two main types of attacks: close-range attacks and long-range attacks. Each of these categories has its own separate mechanics for checking and dealing damage. The systems are written in C#, which is supported by Unity as a scripting language [3], [5], [7].



```
protected void OnTriggerStay(Collider other)
{
    if(m_OwnerCharacter is PlayerCharacter)
    {
        Vector3 hitPlace = other.ClosestPoint(transform.position);
        OnHit?.Invoke(other.gameObject, hitPlace);
    }
    else if (Time.time - m_LastReportTime > m_ReportDelay)
    {
        m_LastReportTime = Time.time;

        Vector3 hitPlace = other.ClosestPoint(transform.position);
        OnHit?.Invoke(other.gameObject, hitPlace);
    }
}

protected void OnTriggerEnter(Collider other)
{
    if (m_OwnerCharacter is PlayerCharacter)
    {
        Vector3 hitPlace = other.ClosestPoint(transform.position);
        OnHit?.Invoke(other.gameObject, hitPlace);
    }
    else if (Time.time - m_LastReportTime > m_ReportDelay)
    {
        m_LastReportTime = Time.time;

        Vector3 hitPlace = other.ClosestPoint(transform.position);
        OnHit?.Invoke(other.gameObject, hitPlace);
    }
}
```

Fig. 5. Code Showing Collision Check for Melee Weapon

Close-Range Attacks:

Close-range attacks include strikes with weapons or other means that require direct contact with the target. This system works as follows:

Collision Check:

- Colliders are invisible geometric shapes attached to objects in the game used to detect collisions (Fig. 4).
- For each frame of the game, it is checked whether the collider of the weapon the player is using collides with the collider of the target object (Fig. 5).

Dealing Damage:

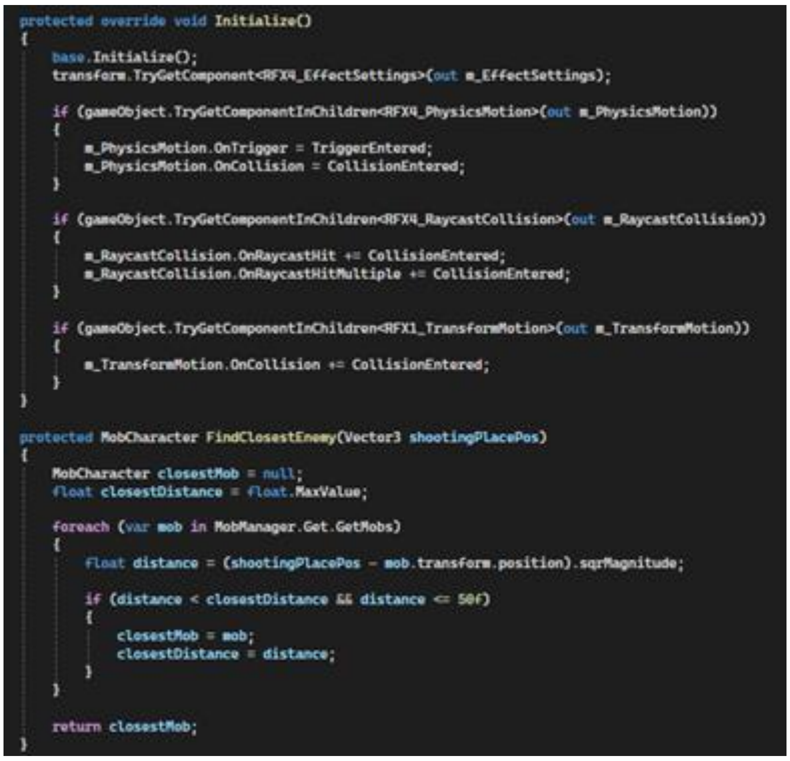
If a collision is detected between the weapon's collider and the target's collider, the corresponding damage is dealt to the target object.

Long-Range Attacks:

Long-range attacks involve the use of weapons or abilities that can hit targets from a distance, such as arrows, bullets, or magical projectiles. This system works as follows:

Projectile Tracking:

- When a long-range projectile is fired, it travels through the game environment until it hits an object (Fig. 6).



```
protected override void Initialize()
{
    base.Initialize();
    transform.TryGetComponent<RFX4_EffectSettings>(out m_EffectSettings);

    if (gameObject.TryGetComponentInChildren<RFX4_PhysicsMotion>(out m_PhysicsMotion))
    {
        m_PhysicsMotion.OnTrigger = TriggerEntered;
        m_PhysicsMotion.OnCollision = CollisionEntered;
    }

    if (gameObject.TryGetComponentInChildren<RFX4_RaycastCollision>(out m_RaycastCollision))
    {
        m_RaycastCollision.OnRaycastHit += CollisionEntered;
        m_RaycastCollision.OnRaycastHitMultiple += CollisionEntered;
    }

    if (gameObject.TryGetComponentInChildren<RFX1_TransformMotion>(out m_TransformMotion))
    {
        m_TransformMotion.OnCollision += CollisionEntered;
    }
}

protected MobCharacter FindClosestEnemy(Vector3 shootingPlacePos)
{
    MobCharacter closestMob = null;
    float closestDistance = float.MaxValue;

    foreach (var mob in MobManager.Get.GetMobs)
    {
        float distance = (shootingPlacePos - mob.transform.position).sqrMagnitude;

        if (distance < closestDistance && distance <= 50f)
        {
            closestMob = mob;
            closestDistance = distance;
        }
    }

    return closestMob;
}
```

Fig. 6. Code Showing Collision Check for Long-Range Weapon

Collision Check:

- When the projectile hits an object, it is checked whether this object is a monster or another target object (Fig. 6).

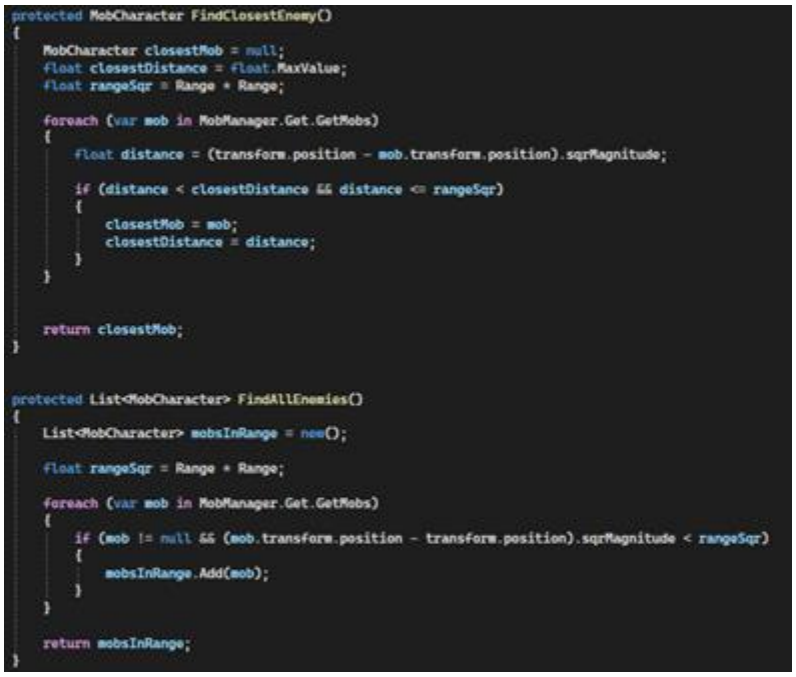
Dealing Damage:

- If the object hit is a monster, it is considered a successful collision, and the corresponding damage is dealt to the monster (Fig. 6).

Tower System:

One of the most important aspects of the tower system is to check whether there are objects within the tower's range so that it can effectively attack enemies. This check involves finding enemies within a certain radius around the tower and calculating the distance from the tower to each enemy. If the enemy is within range, the tower can start attacking.

- Determining the Tower's Range Radius: The tower's range is set by a variable representing the maximum distance at which the tower can detect and attack enemies.
- Iterating Over All Enemies: For each enemy in the game, the distance from the tower's position to the enemy's position is calculated (Fig. 7).
- Distance Calculation: The distance is calculated using the sqrMagnitude method or by using squared distances for more efficient comparisons.



```
protected MobCharacter FindClosestEnemy()
{
    MobCharacter closestMob = null;
    float closestDistance = float.MaxValue;
    float rangeSqr = Range * Range;

    foreach (var mob in MobManager.Get.GetMobs)
    {
        float distance = (transform.position - mob.transform.position).sqrMagnitude;

        if (distance < closestDistance && distance <= rangeSqr)
        {
            closestMob = mob;
            closestDistance = distance;
        }
    }


    return closestMob;
}


protected List<MobCharacter> FindAllEnemies()
{
    List<MobCharacter> mobsInRange = new();

    float rangeSqr = Range * Range;

    foreach (var mob in MobManager.Get.GetMobs)
    {
        if (mob != null && (mob.transform.position - transform.position).sqrMagnitude < rangeSqr)
        {
            mobsInRange.Add(mob);
        }
    }

    return mobsInRange;
}
```

Fig. 7. Methods for Finding the Closest and All Monsters within Tower Range

The sqrMagnitude method in Unity is a property of the Vector2, Vector3, and Vector4 classes. It is used to calculate the square of the length of the vector, which is more efficient than directly calculating its length with magnitude. This method is useful when comparing distances without performing the square root operation.

Pathfinding System for Game Units:

The way the closest path to a given object is determined is done through the A* algorithm, which works on the principle of graph search. It uses a combination of two functions: a function for estimating the distance from the starting point to the current point (often called g) and a function for estimating the distance from the current point to the final goal (called h). Combining these two functions helps the algorithm choose the most efficient path [1].

CONCLUSION

Developing a system for interaction between the player and the game environment in the game "Eternal Vigil" presents numerous challenges and opportunities for innovation in the field of video games. The main advantages of implementing such a system include enhancing realism and player engagement through the use of modern algorithms such as finite state machines and behavior trees. These algorithms allow the creation of intelligent and dynamic behavior of non-player characters (NPCs), leading to a more interactive and interesting game environment. At the same time, the research reveals some disadvantages related to the high technical requirements and complexity of implementing such a system. The development requires significant resources and time for optimization and testing to ensure the system's smooth and seamless operation in real conditions.

REFERENCES

[1] Садулов С., М. Мустафов, К. Шойлекова (2023). Използване на A* алгоритъма в Eternal Vigil Научни трудове на Русенски университет, Том 62, серия 6.1, 52-56, ISSN: 2603-4123

[2] GOLDSTONE, Will. Unity game development essentials. Packt Publishing Ltd, 2009.

[3] HOCKING, Joseph. Unity in action: multiplatform game development in C. Simon and Schuster, 2022.

[4] BLACKMAN, Sue. Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development. Apress, 2013.

[5] MÜLLER-HANNEMANN, Matthias; SCHIRRA, Stefan. Algorithm Engineering. Springer, 2001.

[6] GRAHAM, Ross; MCCABE, Hugh; SHERIDAN, Stephen. Pathfinding in computer games. The ITB Journal, 2003, 4.2: 6.

[7] Lawande, S.R.; Jasmine, G.; Anbarasi, J.; Izhar, L.I. A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games. *Appl. Sci.* 2022, *12*, 5499. https://doi.org/10.3390/app12115499.

[8] AHEARN, Luke. 3D game textures: create professional game art using photoshop. AK Peters/CRC Press, 2016.

[9] S. Wang, Z. Mao, C. Zeng, H. Gong, S. Li and B. Chen, "A new method of virtual reality based on Unity3D,"*2010 18th International Conference on Geoinformatics*, Beijing, China, 2010, pp. 1-5, doi: 10.1109/GEOINFORMATICS.2010.5567608.