Списание „Математика, компютърни науки и образование“ Том 7, кн. 2, 2024
Journal of Mathematics, Computer Science and Education Vol. 7, No. 2, 2024

DOI: 10.54664/UHEW6861

THE STRUGGLES OF CREATING AND SHARING ONE'S STORY THROUGH INDEPENDENT GAME DEVELOPMENT

Yancho Mehmedov Ibrahimov

Abstract: In independent game development, particularly using Unity, various challenges arise throughout the process, from concept to execution. The motivation to create a game often stems from the desire to tell a personal story or express creativity but bringing that vision to life involves technical and artistic hurdles. Technologies like Unity and tools like Blender provide affordable, versatile solutions for modeling, animating, and building game mechanics.
Keywords: Animation, Design, Game Engine, In Game Modules, Modelling

1. Introduction

Art is a diverse expression that can be found in countless forms, from traditional mediums like painting, sculpture, and music to more modern outlets like digital art, interactive installations, and video games.

Video games are being recognized more as a powerful form of art, blending visual design, storytelling, music, and interactive experiences into a cohesive whole package. They offer players not just a chance to observe a narrative but to become an active participant in its unfolding and making choices that shape the story and its outcome. Through immersive worlds, compelling characters, and innovative gameplay, video games push the boundaries of traditional art forms, creating experiences that engage both the mind and emotions on a deep level.

The desire to create a video game often stems from a deep-seated need to share one's stories, experiences, or visions in a way that feels personal and interactive. Independent game developers, or "indie developers", often take this path not just for the love of gaming but also for the desire to break away from conventional narratives and explore ideas that might not fit within the ways of mainstream titles. For many aspiring indie developers, the motivation to create their own game is rooted in the freedom to innovate without the constraints of large studios or commercial pressures. They are driven by the challenge of bringing a world of their own design to life and by the chance to connect with players on a deeper level through interactive storytelling.

2. Computer game development technologies

When it comes to developing a video game, choosing the right game engine is a crucial decision that can significantly impact the final product. Game engines serve as the foundation for creating interactive worlds, providing the tools and technologies necessary for designing graphics, physics, animation, and more. Each game engine comes with its own strengths, features, and limitations, suited to different types of projects and levels of expertise. Here are some of the most popular game engines used in the industry today:

• **Unity**: Known for its versatility, Unity is a favourite among indie developers due to its user-friendly interface and powerful features for both 2D and 3D game development. It supports multiple platforms, has a massive asset store, and uses C# for scripting, making it a great choice for beginners and professionals alike.

• **Unreal Engine**: Developed by Epic Games, Unreal Engine is renowned for its high-quality graphics and realistic rendering capabilities. It's widely used for AAA games but is also accessible to indie developers, thanks to its visual scripting system called Blueprints, which allows for game development without extensive coding knowledge.

• **Godot**: An open-source game engine that's gaining popularity for its lightweight nature and flexibility. Godot supports both 2D and 3D game development and offers its own scripting language called GDScript, which is similar to Python. Its open-source nature allows for a highly customizable experience without any licensing fees.

• **GameMaker Studio**: Ideal for 2D game development, GameMaker Studio is well-suited for beginners due to its drag-and-drop interface, which simplifies the process of creating game logic. It also supports more advanced programming with its proprietary language, GML (GameMaker Language), offering a balance between simplicity and control.

• **CryEngine**: Known for its stunning visuals and realistic environments, CryEngine is often used for creating immersive 3D games. While its graphical capabilities are top-notch, it has a steeper learning curve, making it more suitable for experienced developers looking to create visually demanding games.

• **RPG Maker**: Specializing in creating role-playing games (RPGs), RPG Maker is user-friendly and requires minimal programming knowledge. It features a tile-based system for map creation and a straightforward event scripting interface, making it an excellent choice for those focused on storytelling and classic RPG design.

3. Defining the problems

Game development is a complex process that often comes with a variety of challenges, regardless of the developer's experience level. Some common problems that many developers face during the game creation include Scope Creep (a common issue, where the project grows beyond its original vision as new features and ideas are continuously added), Time and Resource Constraints, Creative Block, Maintaining Motivation, Animation and Art.

For example, if we take a closer look at Animation and Art - creating character models, rigging them, and bringing them to life with animation. This endeavour can be one of the most challenging aspects of game development. Specific problems that can arise during this process can vary between Modelling (Topology, Polygon Count, UV Mapping and Texturing), Rigging (Joint Placement, Weight Painting, Rig Complexity), Animation (Creating Smooth Transitions, Timing and Fluidity) and Integration into Unity.

4. Stages in development

• **Stage 1**

Not knowing how or where to begin in such a broad undertaking can lead to being completely lost in thought, especially for beginners who are not accustomed to any 3D software or have little to no knowledge of it. At this point it is highly recommended to watch other more experienced use said software and learn from them (i.e. videos on social media platforms or software specific sites), that being the case for the majority of self-taught artists and content creators. If there's one lesson from this process, it is that learning new things is invaluable [5, 406-407]. Having the right resources and following the proper guides will carry you to the point where you can freely choose anything you want to create and make it into reality. In fact, going with the "freelancer" route of learning the basics and then following through with self-improving is by far the most efficient way to learn and develop a desired skillset, although it is the most challenging. That's also the reason why there is no shame in using an already built model and modifying it to meet your needs, or completely replicating it from the ground up.

• **Stage 2**

Modelling challenges can significantly impact the final appearance and performance of a character or object. One of the primary issues is ensuring clean and efficient topology (Figure 1), which is crucial for smooth deformation during animation. Poor topology can lead to unnatural bends or distortions when the model is animated. Additionally, optimizing the polygon count is essential—too many polygons can cause performance issues in the game, while too few can result in a lack of detail.

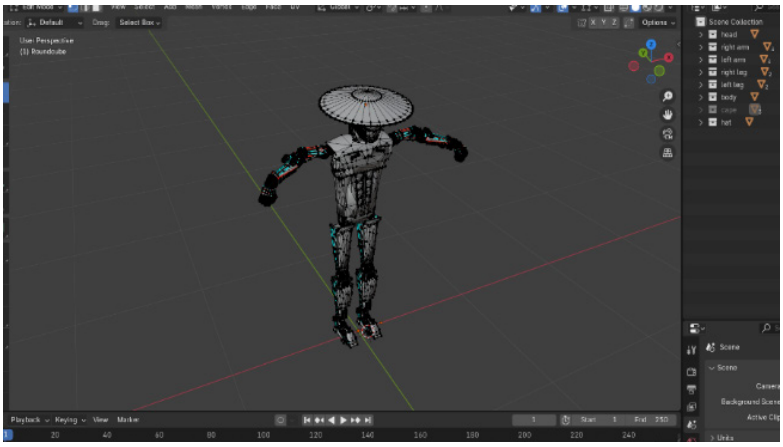

Fig. 1 *The base mesh of a model in Edit View in Blender4.0*

• **Stage**

UV mapping and texturing also present difficulties, as poorly laid out UV maps can cause textures to stretch or misalign, affecting the visual quality of the model. UV mapping involves unwrapping a 3D model onto a 2D plane so that textures can be applied accurately. Poor UV mapping can lead to stretched, distorted, or misaligned textures, making the model appear unpolished. Texturing itself poses additional challenges. Creating textures that fit seamlessly onto the UV map requires skill in both artistry and precision. Developers need to ensure the textures are detailed enough to look good up close but optimized enough to prevent performance slowdowns. Furthermore, achieving consistency across different parts of the model, while dealing with issues like texture tiling and resolution, is crucial for maintaining a cohesive look in-game (Figure 2 and Figure 3).

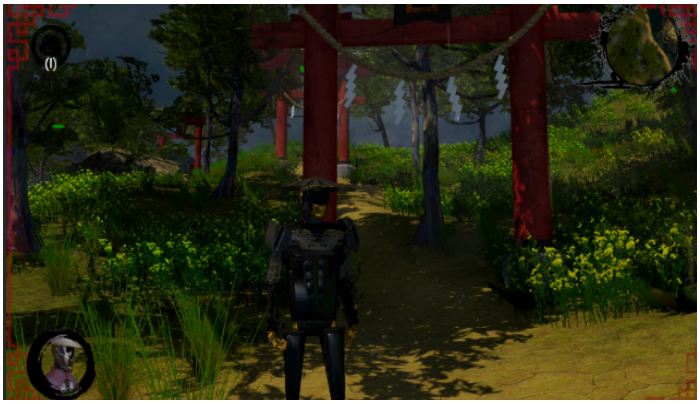

Fig. 2 *The same model from Figure 1 integrated in Unity3D with applied textures*

Fig. 3. *Another scene from Unity3D with the same model and different light settings*

• **Stage 4**

Rigging and animation are crucial stages in bringing a character to life, but they come with their own set of challenges. Rigging involves creating a skeleton (or armature) for the model, defining how each joint and bone controls different parts of the mesh (Figure 4.). Proper joint placement is essential; poorly placed joints can cause awkward deformations when the character moves, leading to unrealistic or broken animations. Weight painting, which determines how much influence each joint has on the surrounding mesh, must be meticulously adjusted to avoid unwanted stretching or collapsing of the model during animation.

Animation as both an art form and popular entertainment has existed for almost a century now [3, 8-9]. Creating natural and fluid motion requires a deep understanding of timing and movement principles. Issues like stiff or robotic movements often arise when keyframes aren't smoothed out, or if the transitions between animations (e.g., from walking to running) are abrupt. In Unity, setting up animation controllers and managing transitions can be complex, and problems such as mismatched loops or incorrect transitions can detract from the fluidity of gameplay.

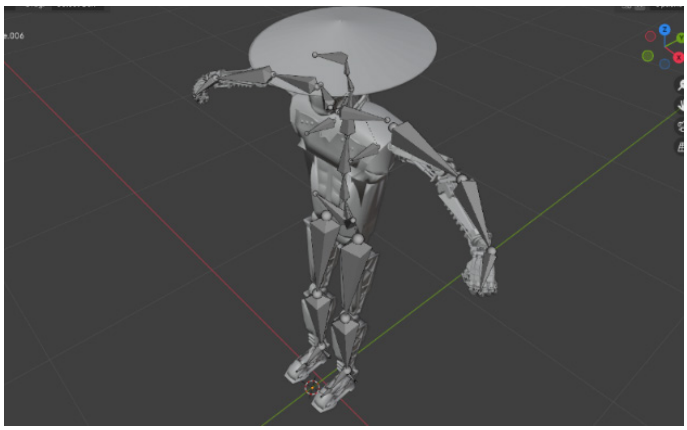

Fig. 4 *The bone structure of the rigged model back in Blender4.0*

5. Choosing the right technology for 3D modelling and game development

Unity provides an excellent entry point into game development, balancing features and functionality with price point. The free version of Unity allows people to experiment, learn, develop, and sell games before committing any of their hard-earned cash [2, 1-2]. Choosing Unity over other game engines is often a matter of its adaptability, ease of use, and strong support network. For indie developers, these qualities mean less time worrying about technical hurdles and more time focused on creativity and bringing their unique vision to life. Here are some of the reasons why Unity might be the best choice:

• Versatility;
• User-Friendly Interface;
• C# Scripting;
• Extensive Asset Store;
• Cross-Platform Support;
• Strong Community and Documentation;
• Regular Updates and Industry Support;

Choosing Blender over other 3D software for creating game assets, specifically for processes like modelling, texturing, rigging, and animation, comes with several compelling advantages:

• Free and Open-Source: Blender is completely free and open-source, making it accessible to indie developers and hobbyists who might not have the budget for expensive licenses.

• Comprehensive Toolset: Blender offers a full suite of tools for modelling, sculpting, UV mapping, texturing, rigging, and animating–all within one software.

• Active Community and Resources: Blender has a large, active community that regularly contributes tutorials, plugins, and custom scripts.

• Frequent Updates and Improvements: Blender is continuously updated with new features and performance improvements, keeping it on par with industry-standard software.

Blender provides several add-ons for architectural and environmental modeling. It's also possible to import three-dimensional environments [4, 167–168].

6. In game modules and interaction between them

In game development, the interaction of core modules (Figure 5.) –Animation, Movement, Physics, and Collision—forms the foundation for a cohesive and engaging gameplay experience. Each module plays a specific role and the synchronization of these systems ensures smooth transitions and realistic interactions, such as a character's movement responding naturally to environmental forces and collisions. Additional systems, like AI for NPC behaviours, Sound for audio feedback, and UI for player information, further enhance gameplay by adding depth and responsiveness. By integrating these modules effectively, Unity enables developers to create dynamic, interactive worlds that feel both immersive and responsive.

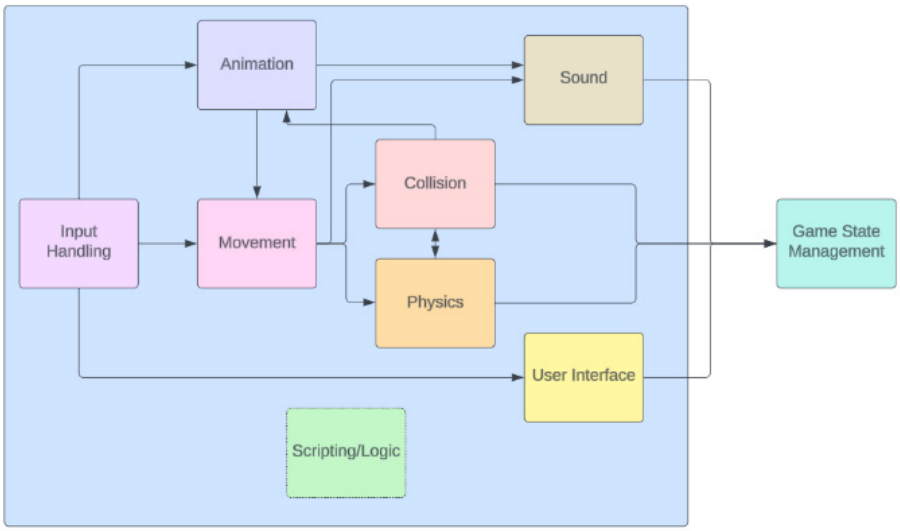

Fig. 5 *Example of how different modules within a game system interact together*

I. **Animation**

• Purpose: Plays visual character or object actions, such as walking, jumping, attacking, or interacting with the environment.

• Interaction: Works closely with Movement to ensure that character actions visually match their actual movement in the game world. Also, in some cases interacts with Physics and Collision.

II. **Movement**

• Purpose: Handles how the player moves through the game world based on input (walking, running, jumping, etc.).

• Interaction: Directly influenced by Input Handling for player control. Triggers Animation (e.g., running, jumping) and relies on Physics for realistic interactions with forces like gravity. Collision ensures that movement stops or changes when the player hits objects like walls or platforms.

III. **Physics**

• Purpose: Simulates real-world physical interactions, including forces such as gravity, friction, and momentum.

• Interaction: Works in tandem with Movement by applying forces such as gravity and drag to characters and objects. Collision detection is a key component of physics, ensuring that objects respond correctly when they touch or overlap.

IV. **Collision**

• Purpose: Detects when objects in the game world encounter each other. Determines whether an interaction, like stopping movement or triggering an event, should occur.

• Interaction: Essential for proper interaction with both Movement and Physics. Collision ensures that characters don't pass through solid objects and triggers responses. Influences Animation, triggering specific animations, such as a landing animation when a player hits the ground.

V. **Input Handling:**

• Purpose: Collects player input from controllers, keyboards, or touchscreens.

• Interaction: Feeds directly into the Movement system, determining how the player character moves, and can also trigger Animations (e.g., jumping or shooting).

VI. **Sound:**

• Purpose: Manages sound effects, music, and audio feedback.

• Interaction: Tied to both Animation and Collision systems, triggering sound effects when a character performs certain actions (e.g., footsteps) or when objects collide (e.g., explosions, impacts).

VII. **UI (User Interface):**

• Purpose: Displays health bars, scores, inventories, and menus.

• Interaction: Frequently interacts with Input Handling to navigate menus or manage inventory systems, and can respond to events triggered by Collision or AI (e.g., when the player takes damage).

VIII. **Game State Management:**

• Purpose: Manages the game's state, including level transitions, game over scenarios, and win conditions.

• Interaction: Works closely with AI, Physics, and Collision to monitor when certain conditions are met (e.g., when an enemy is defeated, or the player reaches a goal).

IX. **Scripting/Logic:**

• Purpose: Controls game logic, such as win conditions, scoring, and in-game events.

• Interaction: Ties all systems together, dictating how they interact, and controlling the flow of the game.

Here is the place to mention an industry-standard term called The 3Cs (camera, control, and character), used in the AAA (triple A) industry in studios such as Ubisoft, EA, Valve and others. It basically states that the core of a player's experience is defined by the balance between the harmonious relationship of the camera, the player character, and the control scheme. If a player feels that the character responds to inputs as expected while the camera seamlessly positions itself at an optimal angle, then the overall experience will be more immersive [1, 17–18].

1. **Conclusions**

The process of creating a game independently is fraught with difficulties–from conceptualizing the initial idea to developing gameplay mechanics and creating visual and auditory elements. These strug-

gles are often compounded by the limited resources and technical skills available to solo developers or small teams. The journey of an indie developer is not merely about crafting a game but also about the determination to share one's vision with the world, regardless of the challenges that lie ahead.

REFERENCES

[1.] D. Baron, (2021). Game Development Patterns with Unity 2021. Packt Publishing, Birmingham, United Kingdom. ISBN 978-1-80020-081-4

[1.] S. Blackman, (2011). Beginning 3D Game Development with Unity. Apress, Los Angeles, California. DOI: https://doi.org/10.1007/978-1-4302-3423-4

[1.] R. Hess, (2011. Tradigital Blender. Focal Press, Oxford, United Kingdom. DOI: https://doi.org/10.1016/b978-0-240-81757-6.00001-x .

[1.] F. Lanzinger, (2022). 3D Game Development with Unity. CRC Press, Boca Raton, Florida. DOI: https://doi.org/10.1201/9780429328725.

[1.] G. Moioli, (2022). Introduction to Blender 3.0, Building a 3D Environment. Apress, Milano, Italy. DOI: https://doi.org/10.1007/978-1-4842-7954-scopefoe0_3.

ИНФОРМАЦИЯ ЗА АВТОРА

Янчо Мехмедов Ибрахимов, студент,
специалност „Компютърни науки“, Факултет „Математика и информатика”,
Великотърновски университет „Св. св. Кирил и Методий”,
e-mail: yancho2002@abv.bg

ABOUT THE AUTHOR

Yancho Mehmedov Ibrahimov, student in Computer science,
Faculty of Mathematics and Informatics, ”St. Cyril and St. Methodius”
University of Veliko Tarnovo, e-mail: yancho2002@abv.bg