

3D for Ancient Rome

Zilin Yuan

C1007159@newcastle.ac.uk

Newcastle University

<https://www.youtube.com/watch?v=UregGPvJKYQ&t=5s>

ABSTRACT

Vindolanda was a Roman auxiliary fortress south of Hadrian's Wall in northern England. Many ancient objects recovered from the archaeological excavations, which is an important key to the daily life of the people of ancient Rome. Every year many people visit Vindolanda to experience the traces of history. But with the development of technology, people can now learn about history in more diverse ways. Educational games are a great way to do this, especially for children, as a game with a colorful and interesting story can be more interesting to them than an incomprehensible historical site. This paper will describe an educational game for children with a background in Vindolanda, discussing how the entertainment and educational aspects can be linked and what tools will be used to help accomplish this. Lastly, it will outline what steps have been completed so far, as well as plans and steps for the project.

KEYWORDS - Vindoland, video games, serious games, education, game engine

INTRODUCTION

History is a subject that everyone studies, in secondary school or even primary school. By studying history and learning from it, people can explore many lessons that they can apply in their present lives. Historic sites are the treasures of mankind and they help mankind to unravel the riddles of history.

1.1 Hadrian's Wall and Vindolada

As one of Britain's best-known ancient monuments, Hadrian's Wall enjoys the status of World Heritage

Site, widespread recognition, huge numbers of visitors and a considerable amount of academic study^[1].

It was a great barrier, helping the Romans to defend themselves against invasion from the north. Hadrian's Wall symbolised the might of the Roman Empire, but it was also its boundary marker, a record of the rise and fall of the ancient Roman Empire.

ROMAN Vindolanda is part of the UNESCO World Heritage Site of Hadrian's Wall, though the site pre-dates the construction of the Wall. A fort was established by the Roman army during the Flavian period of occupation in Roman Britain in the 90s AD, possibly earlier^[2]. It ranks alongside my old third-century hunting ground of Dura-Europos as a site of major importance, in which a snapshot of Roman life has been preserved for posterity. One of these, there have been numerous significant artifacts discovered at the site, which include the Vindolanda writing tablets. The tablets contain correspondence from soldiers (including the Commanding Officer at Vindolanda, Flavius Cerialis) and civilians, providing a fascinating insight into Roman life. Since the first writing tablet was discovered in 1973 over 400 have been found at Vindolanda. The tablets have generated an immense amount of research which has contributed significantly to the knowledge of both military and civilian life in Roman times^[3].

1.2 Serious Games

Technology has impacted almost every aspect of life today and has dramatically expanded access to education^[4]. In contrast to traditional modes of learning, Students tend to use gadgets for playing games rather than reading educational content^[5]. A beautifully illustrated game with an interesting story will be more appealing to students than a paper book.

Educational (or serious) games are gaining attention as attractive and effective tools to create awareness and promoting behavioral change related to sustainability^[6].

The term “Serious Games”, coined in the 70s ^[6], is now widely accepted, Serious games are technology-enhanced simulation games developed for a purpose other than entertainment, such as teaching a specific knowledge or skill^[7]. The expression “serious games” is by itself an oxymoron, as games are defined as a recreational and fun activity, to a sharp contrast of what the word serious conveys^[8].

Games combine ways of knowing, ways of acting, ways of being and ways of caring: situational understandings, effective social practices, strong identities and shared values that make someone an expert.

The expertise might be that of a modern soldier in Full Spectrum Warrior, a zoo operator in Zoo Tycoon, or a world leader in Civilization III. Or it might be expertise in the sophisticated practices of gaming communities, such as those built around Age of Mythology or Civilization III.^[9]

In the game world, learning no longer means being confronted with words and symbols that are separate from the things to which they refer. The inverse square law of gravitation is no longer something that can be understood through equations alone. Instead, students can gain virtual experience of walking on a world with less mass than the Earth, or they can plan a manned space flight - a task that requires an understanding of the changing effects of gravity in different parts of the solar system. In virtual worlds, learners experience the concrete realities that words and symbols describe.^[10]

An online environment that people don't normally experience and that gives them a new way of thinking, not unlike in school where the teacher passes on knowledge to the students, but in games where learning is more about the players themselves and the background story of the game.

From this, we can learn that an educational game, more often than not, chooses to build a virtual world that the player is not normally exposed to and that the

player learns something by 'living' in this world and exploring an unfamiliar one, just as they did as a teenager. In addition to this, the rules of the game that the player must follow are also points that can be explored in educational games, such as in the case of the all-powerful warrior, where the player must refine their skills to win the war, or a simulated firefighting game where the player follows the rules of the game to win. Of course, there are games set in modern cities, but obviously, an unfamiliar city is more appealing to the player.

BACKGROUND and RELATE WORK

Generally speaking, serious games are classified into different genres based on their narrative design and core mechanics, such as strategy games, simulation games, action games, etc. The main goal of the game is to teach something to the player and the learning by design takes place through playing the game itself.

Designing a serious game that blends fun with a pedagogical purpose can be difficult. Before designing a game, it is important to first consider some of the following issues.

▪ What about?

The theme of the game must be clear as to what kind of educational content it wants to bring to the player through this game.

Many serious games fall into the category of simulations of all kinds. The classic video game 'Oregon Trail' was originally developed in the early 1970s and was designed to give players a realistic feel for life in the 19th century. Complex role-playing games, such as the 'Reacting to the Past' series, are designed to allow students to investigate and explore key historical events, for example in the scenario 'Greenwich Village in 1913. Suffrage, Labour and the New Woman'^[11] in an in-depth and meaningful way. Scientific games, such as Fold It, encourage players to investigate complex scientific phenomena, such as the folding of proteins^[12].

For a game based on a historical site, the central point is how to communicate the historical story of the prototype to the player.

▪ Who for?

Understanding the end user of a program is an important consideration in the design of all programs. Full Spectrum Warrior (Pandemic Studios, for PC and Xbox), is a video game based on a training simulation for the US Army. In order to survive and win the game, the player must learn to think and act like a modern professional soldier. Throughout the game, players do not directly control any of the fireteam members but gives orders by operating the controller, as well as consult GPS devices and radio support and communicate with their commanders in the rear. The instruction manual that accompanies the game makes it clear from the outset that if players are to play the game successfully they must embrace the values, identity and mentality of a professional soldier.^[9]

The work was proposed as a children's project, mainly to make Vindolanda accessible to the players, and it was a goal of the project for the children to know what was in Vindolanda and what had happened through the game. The project is also going to tend to be easier to start playing and to make the historical story more interesting in the game.

- How is it going to be played?

The platform on which the game is released is also important. Different gaming platforms have different tendencies, but one of the major advantages of choosing a pc platform to publish your game is that more players can start playing directly on their computers

- When is it play?

A game can be played for minutes or hours, and the length of time it is played is an important factor. Although research shows that games do not affect the mental health of players in general, and that they may have a positive effect on young children,^[13] the duration of play in this project is not too long.

- Why a game?

This is a fundamental issue for serious games, and how to combine games and education is something to think through before making serious games.

In this paper studied, to show what Vindolanda looks like to the player, the choice of 3D visuals is a good choice. Digital and printed models may remove object authenticity, but they do provide direct encounters

with heritage and archaeological science whilst preserving the archaeological record^[14].

Unity3d

Unity 3D, also known as Unity, is a comprehensive multi-platform game development tool developed by Unity Technologies that allows players to easily create interactive content such as 3D video games, architectural visualisations, real-time 3D animations and other types of interactive content.^[15]

There are countless commercial and free game engines in the industry, with the most iconic commercial game engines including Unreal, CryENGINE, Havok Physics, Game Bryo, Source engine and many more. However, all of these game engines are expensive, making game development significantly more expensive. Unity's slogan is "Democratising Development" and it offers a great game engine that anyone can easily develop with no price restrictions. It both gives access to the necessary elements for virtual world building and compels developers to adopt norms derived from the game and tech industries in which it is enmeshed.^[16]

Game developers can develop on a variety of platforms. Once a game has been created, no modifications are required and it can be published directly to popular mainstream platforms with just a click.

Unity 3D games can be published on platforms such as Windows, Linux, MacOS X, iOS, Android, Xbox 360, PS3 and the web. Cross-platform development can save game developers a great deal of time.^[17]

In the past, developers had to consider differences between platforms, such as screen size, operating style, hardware conditions, etc. Unity 3D solves this problem almost perfectly for developers and will greatly reduce unnecessary hassles in the porting process.

- Integrated Editing

Unity 3D's user interface features visual editing, a detailed property editor and dynamic game preview. Unity 3D's innovative visualisation mode allows game developers to easily build interactive experiences and modify parameter values in real time while the game is running, saving a lot of time in game development.

- Resource import

Unity 3D supports almost all major 3D formats, such as 3ds Max, Maya, Blender, etc. Textured materials are automatically converted to U3D format and work with most relevant applications.

- One-click deployment

Unity 3D can be developed and deployed across multiple platforms with a single click, allowing developers to render their work across multiple platforms.

- Scripting Language

Unity 3D integrates with the MonoDeveloper compiler platform and supports 3 scripting languages: C#, JavaScript and Boo, with C# and JavaScript being the most common scripting languages used in game development.

Blender

Blender is an open-source, cross-platform, all-in-one 3D animation software that provides a range of short animation solutions from modeling, animation, materials, rendering, to audio processing and video editing. It has a wide range of user interfaces for different tasks and advanced film and video solutions such as green screen keying, camera back tracking, masking and post node compositing. There are also built-in cartoon strokes (FreeStyle) and a GPU-based Cycles renderer. A wide range of third party renderers is supported with Python as a built-in script.

Blender offers a complete shortcut to immersive modeling BMesh systems. With support for n-edge, professional modeling tools, the ability to create industrial-grade products and creature models. Together with the sculpting topology tool, you can quickly create realistic models of people, animals and more. As it is open source software, it conveniently offers cooperation and collaboration with other 3D software.

UV unfolding in blender is the fastest and most intelligent solution in 3D software. This includes one-click UV unfolding and multi-layer UV support, which is a great advantage. This, together with seamless material drawing, increases not only efficiency but also quality.

DESIGN and IMPLEMENTATION

The project starts with the construction of a 3D Vindolanda, which is first sketched out on paper, with the layout of the houses and some ancient Roman styles. The player takes on the role of a man who accidentally arrives in ancient Vindolanda and interacts with the civilians and soldiers living in Vindolanda, helping them with their tasks and perhaps some battles.

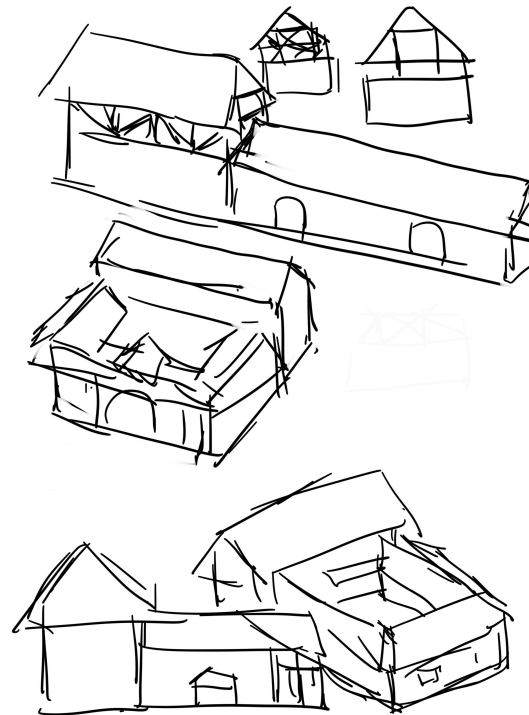


Fig.1 Modeling sketches of the ancient Vindolanda

Blender modelling

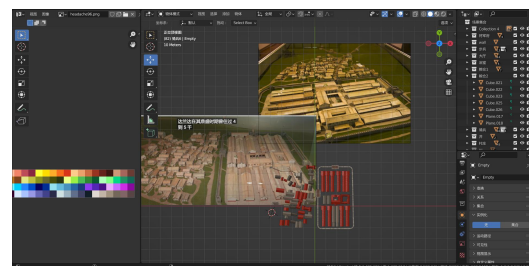


Fig.2 Blender

The image background set in blender can be limited to a few views, which can be used as a good reference in the modelling process without blocking the modelling. The quadrilateral is edited to change its size and shape to create the appearance of the ancient vindolanda. The built-in modifiers in blender are a great aid to modelling, such as mirroring, surface subdivision, and

simple deformation. Arrays can be used to create a Roman-style roof by creating a few small tiles and using the modifier to lay them over a large area of the model's roof.

The shader maps of the models can be used in a single swatch, bringing together the UV maps of all models in a single image, and models sharing a single material sphere can be a good way to reduce resource usage.



Fig.3 Model

Character movement

In a virtual reality 3d game, the first step is to get the character moving. A navigation mesh so-called NavMesh is a data structure for route planning in a virtual environment and is particularly useful in game development. It consists of a set of convex polygons representing map areas; the edges of the polygons are annotated with additional connection information showing which areas the game character can traverse. The AI characters in the game, which include more than just the player, often need to find an optimal path to reach their destination. There are a very large number of pathfinding algorithms to choose from.

The initial Unity navigation system was very imperfect, as it could only statically bake the walkable area of the scene map and had to save the scene's NavMesh data locally, making it difficult to calculate dynamically at runtime; this made few developers willing to try Unity's built-in navigation functionality and turned to the AStar pathfinding algorithm, which is based on traversing a grid to calculate walk weights. The complexity of the algorithm is relatively high and is influenced by the density of the grid, the size of the map and the length of the route.

Unity's NavMesh uses a corner point algorithm, and the baked NavMesh area only has vertices at the edges of obstacles and the edges of the plane, not uniformly

across the plane like AStar; if it is a plane without any obstacles, there will only be a few vertices at the edges of the plane, and the algorithm is relatively efficient and does not change significantly as the map gets larger. There is no significant change in algorithmic complexity as the map becomes larger.

Conversely, the disadvantage of NavMesh is also the advantage of AStar, which is that it is difficult to guarantee an optimal solution for pathfinding and is more often used for the AI to be able to calculate a faster path around the obstacles towards the target.

For static maps where the scene remains the same, Unity's original NavMesh was adequate. In addition, by setting up a navmeshagent component for the characters in the game, the movement of the characters and the interaction between them can be controlled visually through the component

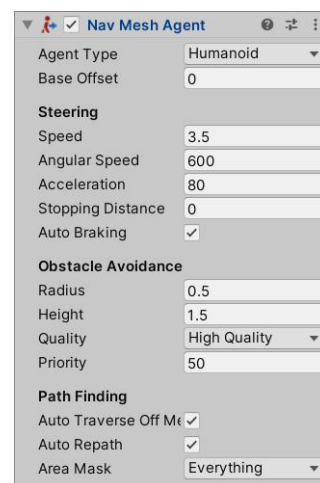


Fig.4 NavMeshAgent for character moving

Player and Enemy Setting

The main functions that the player needs to perform are movement and attack. Player movement is achieved by creating an event in the mouse controller that requires a Vector3 variable, obtaining the player's gameobject, and sending the mouse click point as input to the agent as the target point via the NavMeshAgent component mounted on the player's body, allowing for code free movement.

If you wanted to create a patrolling NPC that moved randomly on its own, found the player and then caught up with them, you'd often need to handle all sorts of events and state switching, write all sorts of switches/cases and if/else, so we'd probably be

dealing with finite state machines all the time, but we might not be aware of it. If you can abstract the business model into a finite state machine, then the code is particularly logical and structured

Finite-state-machine is a tool for modeling the behaviour of an object. Its role is primarily to describe the sequence of states that an object experiences during its life cycle and how it responds to various events from the outside world. It can be applied at various levels, such as in hardware design, compiler design, and when programming the implementation of various concrete business logics.

Controlling the simple random movement of enemies through a simple state machine.

- **State.** A state machine must contain at least two states. An aggressive enemy has at least one of the following states: attack, stationary and dead, and in each state the enemy has different events to control the action.
- **Event.** An event is a trigger condition or a command to perform an action. For enemy npc's events such as random movement, pause and chasing the player
- **Action.** When an event occurs, an action is performed. For example, if the event is "attack player", the action is "swing sword". For programming purposes, an Action usually corresponds to a function.
- **Transition.** That is, a change from one state to another. For example, "patrolling" to chasing a player is a transition.

When a condition or event is met, such as the NPC's blood level going to zero, then the NPC will enter the state of death, destroy the gameobject and disappear from the map after the death animation has been played, and turn off the other components of the NPC during this time, making it impossible for the player to interact with them again. the player can no longer interact with them. When building finite-state-machine, it is important to avoid treating a "program action" as a "state". So how do you distinguish between 'actions' and 'states'? "An "action" is unstable, even if not triggered by a condition, and ends once it has been executed; whereas a "state" is relatively stable, if not triggered by an external condition. If no external conditions are triggered, a

state will continue to exist.

Enemies are divided into standing and patrolling modes, but when they spot the player, they all have to choose to run towards the player and start attacking. `physics.OverlapSphere()` enables retrieving every collider within a set visual range, using `foreach` to find a player whose name is collider, returns a bool value, and enters the chase state.

Implementing a character's attack is done by determining whether the character is within the attacker's attack range. The player selects the enemy by mouse click and the enemy is queried for a collision body with the name player using the `Physics.OverlapSphere()` function. When the player and the enemy are in the attack range, the attack animation will start, using the `LookAt()` function to ensure that the attacker turns to the attacker first. By creating an animation event in the attack animation for the function that deducts blood on hit, it is possible to achieve the effect of deducting blood when the animation plays to the attack.

At the same time, if you want the player not to take damage from an attack after the player has left the front of the enemy when the enemy is attacking, you have to restrict the player's damage function so that when the player's coordinates are not in a sector directly in front of the enemy, the damage function of the attack is not performed. Dot can get the dot product of two vectors, if the normalized vectors point in exactly the same direction Dot returns 1, if the exact opposite is -1, perpendicular to each other returns 0, using this method can calculate the angle between the player and the enemy, when the value is less than 0.5, you can restrict the enemy did not attack the player, do not perform the damage function .

Game management and Singleton

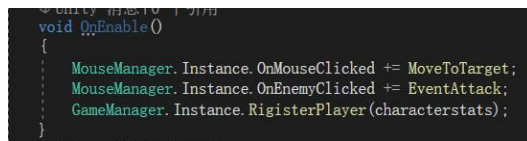
Singleton pattern is defined in GoF as confirming that the class has only one object and providing a global method to obtain that object. It requires the support of the programming language when implemented. As long as it has static class properties, static class methods, and a redefined class builder access hierarchy

Singleton involved in the following description of the roles involved in the project.

- Can generate one unique object class and a "global method" is provided to allow the outside world to easily access the unique object.
- Common to set the only class object to be a "static class attribute". It is customary to use Instance as the name of the global static method through which the "static class property" may be obtained.

In Unity 3D, when switching scenes, all game objects in the previous game scene are destroyed by default, so the Player objects created in the scene will be destroyed. You can use the `DontDestroyOnLoad ()` method to keep the Player objects from being destroyed when switching scenes, added to the `Awake ()` method of the Manager method in the `Awake ()` method of the Manager class.

As shown in the diagram, by inheriting from singleton's script as the controller in the game, functions that are often called in the implementation of the game can be easily called from the controller's script.

A screenshot of a C# script in Unity, showing a Singleton pattern implementation. The code includes a static `Instance` property and a `GetInstance()` method. The `OnEnable()` method is overridden, and it calls `MouseManager.Instance.OnMouseClicked += MoveToTarget;`, `MouseManager.Instance.OnEnemyClicked += EventAttack;`, and `GameManager.Instance.RegisterPlayer(characterstats);`.

```
using UnityEngine;
using System.Collections;

public class Singleton : MonoBehaviour
{
    static Singleton Instance;

    void Awake()
    {
        if (Instance == null)
        {
            Instance = this;
        }
    }

    static Singleton GetInstance()
    {
        return Instance;
    }
}
```

Fig.5 Using Singleton

Save and Load Data

Storing data in a game becomes necessary when the player wants to quit the game, but the next time they enter the game they want to pick up where they left off. There are various ways to implement saving and storing data in unity. `PlayerPrefs` is a data structure that comes with Unity and is located under the `UnityEngine` namespace. It allows access to 3 types of data: integer, floating point and string. `PlayerPrefs` stores all data internally as a dictionary, i.e. each piece of data is a Key-Value Pair. The Key refers to the name of the data, which must be of type string and cannot be repeated, while the Value stores the data value itself and can be of any object type without restriction. In the project, we also used the `JsonUtility` method, using `Tostring` to serialize the stored data into

string types, and then storing the data on the computer's hard drive using `PlayerPrefs`, and when we need to call it, we can retrieve the data by key, and use `JsonUtility's FromJsonOverwrite` restores the data to its original form and assigns it back

In the game, the singleton script is called to get the player's calculated life values, or items, and their names are transferred as keys to the save script.

ScriptableObject 和各种永久保存的数据

A `ScriptableObject` is a data container that can be used to store large amounts of data. the main use of a `ScriptableObject` object is to reduce the memory footprint of a project by storing data in it and avoiding duplication of values.

When we are using a prefab with an inherited `MonoBehaviour` script that holds a constant amount of data, each time we instantiate the prefab we make a copy of the data. Using a `ScriptableObject` object to store the data and then access the data in the prefab by reference avoids the need to copy the data in memory. Like `MonoBehaviour`, `ScriptableObject` inherits from the Unity base class object, but unlike `MonoBehaviour`, `ScriptableObject` cannot be associated with a `GameObject` pair; instead, it is usually saved as an Assets resource.

Characters' Data

In editor mode we can save data to a `ScriptableObject` at edit and run time, as saving a `ScriptableObject` requires the use of an editor space script, but in development mode, you cannot use a `ScriptableObject` to save data, though you can use data saved from the `ScriptableObject` resources to save data.

The player can switch weapons when attacking, or choose to go empty-handed, and gain more experience as the battle progresses, which then improves his abilities. Behind the implementation of these features, the player's (and NPCs too, different NPC's can share the same data) attack power, blood, etc. can be saved as `ScriptableObject` and can be called up by the calling character at any time in unity. When one type of enemy uses the same data, one enemy killed may cause all other enemies to die, and sharing the same `ScriptableObject` data will cause their blood levels to

change simultaneously. You can make a copy of the ScriptableObject data set to the enemy NPC at the beginning of the game by Instantiate() and change the enemy data in real-time in the copied data to differentiate between a single enemy and a group of enemies.

Backpack and Items

In addition to the basic storage of character data, the game also creates ScriptableObjects for the backpacks, items in the backpacks, quests, including player and NPC conversations to be stored in unity. All files will have a proprietary item name that can be displayed with the GUI, and can also be transferred in as a key when implementing save and read data.

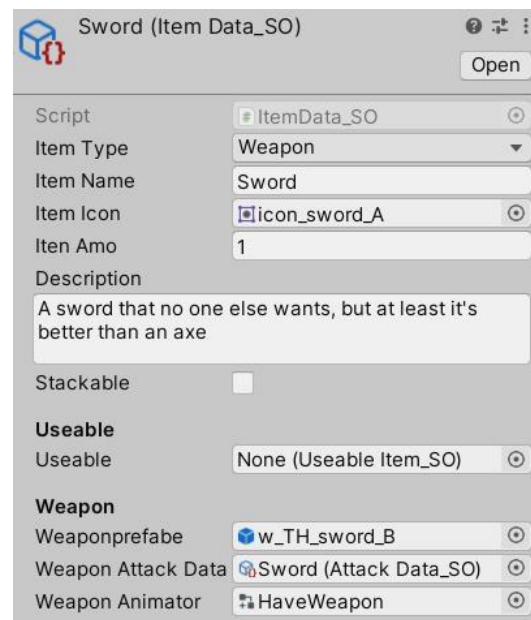


Fig.6 Using ScriptableObjects to save item

In addition to the backpack, the player will also have a shortcut bar and a weapon bar. Setting up different ScriptableObject class files for the three backpacks allows for good organization of the backpack and makes it easier to implement drag-and-drop judgments for items later on. In the ScriptableObject class file for the items, in addition to the basic name, quantity, and picture. The items of the available type and the items of the weapon type also inherit separate ScriptableObject class scripts and call the ScriptableObject file.

Some backpack items, such as weapons, can be recorded in the same way as their attack data, and the prefab without any plugins. When the player puts a

weapon into the weapon box, the corresponding weapon prefab is generated in the player's right hand position, and the player's animation is switched to the corresponding animation in the weapon file. In order to prevent the animation of the player holding the weapon from going wrong, the rotation of the weapon prefab should be tested on the player's body in advance.

Dialogue and mission system

The dialogue system in this project is also implemented by calling the data saved in the ScriptableObject class file. Two separate dialogue classes are created to script the ScriptableObject class for the dialogue system, the dialogue script for the NPC and the answer script for the player. Determine whether or not to take a quest by adding a quest component to the dialogue component and determining which answer the player clicks (Get Book Value while player click the answer YES button).

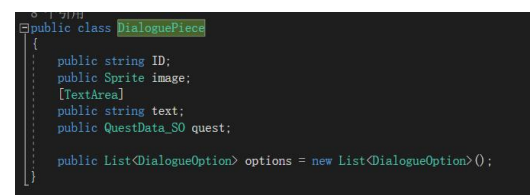


Fig.7 Dialogue system

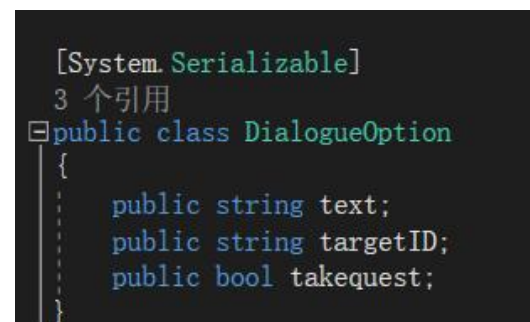


Fig.8 Set mission

Using List<> to create lists so that each dialogue file can contain more than one sentence.

The quests can be submitted by the number of items in the quest file to enable the player to receive rewards as well as submit the items. The quest items that need to be submitted to the NPCs can be deleted from the player's backpack by setting a negative number in the quantity. When submitting quests, a function is required to determine the number of items in the player's backpack and shortcut bar to ensure that the

player collects enough quest items to submit some of the items placed on the shortcut bar.

```

public void GiveRewards()
{
    foreach (var reward in reward)
    {
        if (reward.amount < 0)
        {
            int requireCount = Mathf.Abs(reward.amount);
            if (InventoryManager.Instance.QuestItemInBag(reward.itemData) != null)
            {
                if (InventoryManager.Instance.QuestItemInBag(reward.itemData).amount <= requireCount)
                {
                    requireCount -= InventoryManager.Instance.QuestItemInBag(reward.itemData).amount;
                    InventoryManager.Instance.QuestItemInBag(reward.itemData).amount = 0;
                    if (InventoryManager.Instance.QuestItemAction(reward.itemData) != null)
                    {
                        InventoryManager.Instance.QuestItemAction(reward.itemData).amount -= requireCount;
                    }
                }
                else
                {
                    InventoryManager.Instance.QuestItemInBag(reward.itemData).amount -= requireCount;
                }
            }
            else
            {
                InventoryManager.Instance.QuestItemAction(reward.itemData).amount -= requireCount;
            }
        }
        else
        {
            InventoryManager.Instance.InventoryData.AddItem(reward.itemData, reward.amount);
        }
    }
}

```

Fig.9 Find out mission item

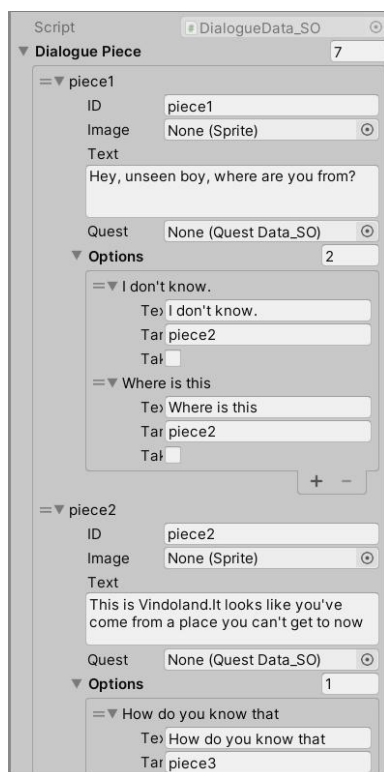


Fig.10 Creat a dialogue file

Game GUI

The main part of the UI for this project is the implementation of the backpack bar drag and drop, the swapping of items and the introduction of items.

The drag-and-drop function for backpack items is achieved by clicking the mouse, accepting the grid to which the mouseup and mousedown positions point, obtaining information about the original item in the grid and updating the grid's information to achieve the effect of moving the item.

Use EventSystem to determine the pointer

position, EventSystem.current.IsPointerOverGameObject() to determine if the mouse has touched the GUI gameobject. 's RectTransform contains information about the position, size, anchor point and axis of the rectangle. The player's backpack is checked with a for loop of backpack data, using RectTransformUtility.RectangleContainsScreenPoint to determine whether the player's mouse has moved within the backpack's grid range. And when swapping items, when the item originally existed in the grid, that is, when the UI picture of the item is displayed, EventSystem.current.IsPointerOverGameObject()

returns the gameobject of the picture, so when moving the mouse to get the gameobject of the picture, it needs to return the parent level of the gameobject.

When moving objects directly, the UI image will appear under the UI image of the grid, you can create a new Canvas and place it at the bottom (unity's default rendering order), and when dragging objects, place the clicked grid under this Canvas so that the objects are always at the top of the UI.

To achieve the drag-and-drop movement of the backpack bar, the IDragHandler interface of EventSystems is used. The drag and drop effect is achieved using RectTransform.anchoredPosition, which gets the center point of the panel component. By adding the distance that changes when the mouse is moved in the OnDrag function, the panel can follow the mouse movement and achieve drag and drop effect. In addition, in order to match the screen of the game, you need to divide the size of the Canvas in the function.

EVALUATION

In this work the creation and construction of the game scenario was implemented, allowing the character to walk around the city, talk to NPCs, accept quests and get paid for them, have fights with NPCs, show the blood levels of both sides and some NPCs drop rewards when they die. The backpack function and the ability to switch between weapons is implemented. In the backpack the player can drag and drop objects and drag them to the shortcut bar. Player can save and load game data.

In this way, these features build a game large enough for the player to navigate, and the player is free to explore in this 3D Vindolanda.

The game also uses a shaper graph plug-in to give the game a more diverse and visually positive rendering. This includes the creation of a glowing material sphere to show the player's position when it is obscured by another object.

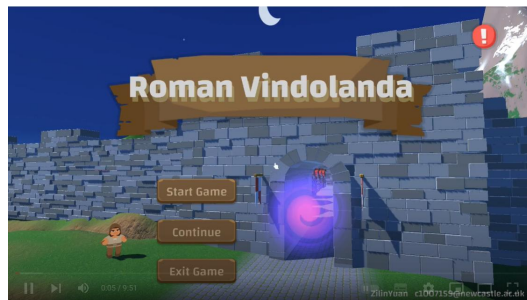


Fig.11 Game start Interface

In addition, there are several things that could be improved in the game. For example, the number of NPCs and more dialogue. Writing dialogue is a huge task, including the logic for the quest system, setting up different dialogue files and quest files for the NPCs, and creating more quest props and items. This project has only completed the initial framework, and there is no time to add more complex and gameplay-rich content to the game.

In addition to these additions to the game, there are some other features that would complete the game. Take for example the audio system. Adding music to the game, with scene changes and sound effects when the player walks or attacks, would make the game more complete and more interesting, and the sound effects that accompany the game content would enhance the experience.

The original plan was also to use the TimeLine plug-in, with animations, to make the game more informative and to allow the player to be guided through the game world with these little animations. However, due to time constraints, this part was abandoned.

Finally, one of the things that this work is always trying to do, and perhaps never will be enough, is to try and convey as much of the ancient Vindolanda as possible. This is based on my knowledge of Roman

history, including the architecture of Vindolanda, and the style of clothes and shoes. The large number of wooden planks found in Vindolanda can also be used as good educational material for writing up information for the player's conversations with the NPCs. There are also many archaeological finds in Vindolanda, like chessboards and wooden toilet mats that can be used as in-game messages to the player.

CONCLUSIONS

Overall, the project implements a presentation of ancient Vindolanda, through which the player can navigate in Vindolanda and talk to NPCs. There is also a lot of room for enrichment of the game.

This project has also given me a deeper appreciation for the creation of a game, including the creation of models, and a deeper understanding of the use of Blender. This will be a valuable experience for me in my future work.

References

- [1] Nesbitt, C.; Tolia-Kelly, D. (2009). Hadrian's Wall: Embodied archaeologies of the linear monument. *Journal of Social Archaeology*, 9(3), 368–390.
- [2] STOCKS, CLAIRE, BIRLEY, BARBARA AND COLLINS, ROB(2018). Stories from the Frontier: Linking Past and Present at Vindolanda through Digital Gameplay. *thersites* 8 (2018),97-109.
- [3] Naomi Kinghorn; Ken Willis (2008). Valuing the components of an archaeological site: An application of Choice Experiment to Vindolanda, Hadrian's Wall. , 9(2), 117–124.
- [4] Khan N, Muhammad K, Hussain T, Nasir M, Munsif M, Imran AS, Sajjad M(2021). An Adaptive Game-Based Learning Strategy for Children Road Safety Education and Practice in Virtual Space. *Sensors*. 2021; 21(11):3661.
- [5] Nilam Sri Anggraheni, Nurul Hidayah, Ayu Nur Shawmi(2019). *AL IBTIDA: JURNAL PENDIDIKAN GURU MI*, Vol 6 (1): 49-62
- [6] Roba J, Kuppens T, Janssens L, Smeets A, Manshoven S and Struyven K(2021). Serious Games in Secondary Education to Introduce Circular Economy: Experiences With the Game EcoCEO. *Front. Sustain*.
- [7] Ju-Hui Wu, Je-Kang Du & Chen-Yi Lee (2021) Development and questionnaire-based evaluation of virtual dental clinic: a serious game for training dental students, *Medical Education* Online, 26:1.
- [8] O. Heidmann(2015).How to create a serious game?:*SGVol* 2(6),1-5.
- [9] Shaffer, D. W.; Squire, K. R.; Halverson, R.; Gee, J. P. (2005). Video Games and the Future of Learning. *Phi Delta Kappan*, 87(2), 105–111.
- [10]W. Westera; R.J. Nadolski; H.G.K. Hummel; I.G.J.H. Wopereis (2008). Serious games for higher education: a framework for reducing design complexity. , 24(5), 420–432.
- [11] Kelly, K.A. A yearlong general education course using “Reacting to the Past” pedagogy to explore democratic practice. *Int. J. Learn*. **2009**, 16, 147–155.
- [12] Ardito, Gerald, and Betül Czerkawski. (2021). "The Development of Autonomous Student Learning Networks: Patterns of Interactions in an Open World Learning Environment for Teachers Exploring Teaching with and through Computer Science" *Sustainability* 13, no. 16: 8696.
- [13] Kovess-Masfety, Viviane; Keyes, Katherine; Hamilton, Ava; Hanson, Gregory; Bitfoi, Adina; Golitz, Dietmar; Koç, Ceren; Kuijpers, Rowella; Lesinskiene, Sigita; Mihova, Zlatka; Otten, Roy; Fermanian, Christophe; Pez, Ondine (2016). Is time spent playing video games associated with mental health, cognitive and social skills in young children?. *Social Psychiatry and Psychiatric Epidemiology*, 51(3), 349–357.
- [14] Williams, R. & Thompson, T. & Orr, C. & Birley, A. & Taylor, G., (2019) “3D Imaging as a Public Engagement Tool: Investigating an Ox Cranium Used in Target Practice at Vindolanda”, *Theoretical Roman Archaeology Journal* 2(1), p.2.
- [15] Li Zhen, Yu Haidong. A study on building a virtual campus roaming system based on Unity3D technology [J]. *Computer Programming Skills and Maintenance*, 2021(07):144-146.]
- [16]Foxman, Maxwell (2019). United We Stand: Platforms, Tools and Innovation With the Unity Game Engine. *Social Media + Society*, 5(4), >
- [17]Nicoll, Benjamin; Keogh, Brendan (2019). The Unity Game Engine and the Circuits of Cultural Software || , 10.1007/978-3-030-25012-6(), –.