Proceedings of CONF-MLA 2024 Workshop: Securing the Future: Empowering Cyber Defense with Machine Learning and Deep Learning
DOI: 10.54254/2755-2721/110/2024MELB0110

Exploring the Application of Game Engine in Creating 2D Fighting Game

Zhongyu Chang

KangChiao Xi'an Qu Jiang, Xi'an, China

130184@stu.kcisxa.org.cn

Abstract. With the burgeoning growth of the digital entertainment industry, 2D side-scrolling fighting games, as a classic genre, are confronted with the dual challenges of innovation and technological renewal. To address these challenges, this study aims to delve into the development techniques and methodologies of 2D side-scrolling fighting games, with a particular focus on leveraging game engines to enhance the overall quality and player experience of the games. Utilizing a combination of literature review, case study, and empirical research, this study conducts a thorough assessment of current mainstream fighting game engines. By analyzing successful cases of 2D side-scrolling fighting games both domestically and internationally, this research endeavors to distill an effective development process and key technical points to guide game developers in making more informed decisions during the design and implementation phases. The findings of this study not only contribute to improving the playability and market competitiveness of games but also offer new perspectives and strategies for the sustainable development of the gaming industry.

Keywords: 2D side-scrolling fighting games, game engines, technical assessment.

1. Introduction

With the rapid development of mobile internet technology, the mobile gaming market has shown robust growth [1,2]. Among these games, 2D side-scrolling fighting games have gained significant popularity due to their unique gameplay experience and operational appeal. These games attract a large number of loyal players with their exquisite graphics, rich character design, and smooth control. Characterized by being easy to pick up but challenging to master, they cater to both casual gamers and hardcore players seeking a higher level of competitive experience.

Despite notable advancements in China's gaming industry, the development of 2D side-scrolling fighting games still faces numerous challenges. Compared to international standards, domestic games in this genre have room for improvement in terms of quality and technical sophistication. Areas that require further research and development include technological innovation, user experience optimization, and originality of game content. 2D side-scrolling fighting games have established a presence in the domestic gaming market, but they lag behind their international counterparts in several respects [3]. On one hand, Chinese game developers lack strong technical foundations, particularly in selecting and utilizing game engines effectively. Game engines, as the core technological support for game development, significantly impact a game's visual presentation and performance efficiency. Although popular game engines like Unity and Unreal Engine offer robust support for 2D game

DOI: 10.54254/2755-2721/110/2024MELB0110

development, Chinese developers often fail to fully exploit their potential [4]. On the other hand, a lack of innovation in game content is also a significant issue. Many 2D side-scrolling fighting games feature conservative gameplay designs that struggle to engage younger players. As the player base diversifies, games must continuously innovate to remain competitive, which encompasses not only gameplay but also narrative, character design, and interactive elements.

To address these challenges, this study aims to explore the development technologies and methodologies for 2D side-scrolling fighting games, focusing on leveraging game engines to enhance game quality and playability. The goal is to provide Chinese game developers with a systematic guidance framework to overcome technical obstacles and advance the development of domestic 2D side-scrolling fighting games.

Specifically, this research will employ literature reviews, case studies, and empirical research to comprehensively evaluate current mainstream fighting game engines. Drawing from experiences with outstanding domestic and international works, this paper aims to distill a set of development processes and technical key points suitable for 2D side-scrolling fighting games.

2. Design and implement of 2D fighting game

2.1. Basic settings of the game.

Using the pixel style in this game, the player first need to make the character's standing animation and collision area. set the viewport size to 1/3 of the original size, change the stretch mode to canvas items, add tile map and tile set.

Figure 1 is about the selection of animation frames. Draw a map with pictures. Figure 2 shows the mapping steps it helps players draw maps faster and more easily. Add sprite 2D nodes, and create a standing animation after editing the animation frame, which is reflected by changing the speed of the character. After that, the player needs instantiate the player scene and add different buttons to control different animations in the game [5]. Create variables in the scene panel to play the corresponding animation. For example, according to the player input direction, set the picture horizontal flip, to achieve the left movement jump [6,7].

Figure 1. In-game character action material (Photo/Picture credit : Original)

DOI: 10.54254/2755-2721/110/2024MELB0110

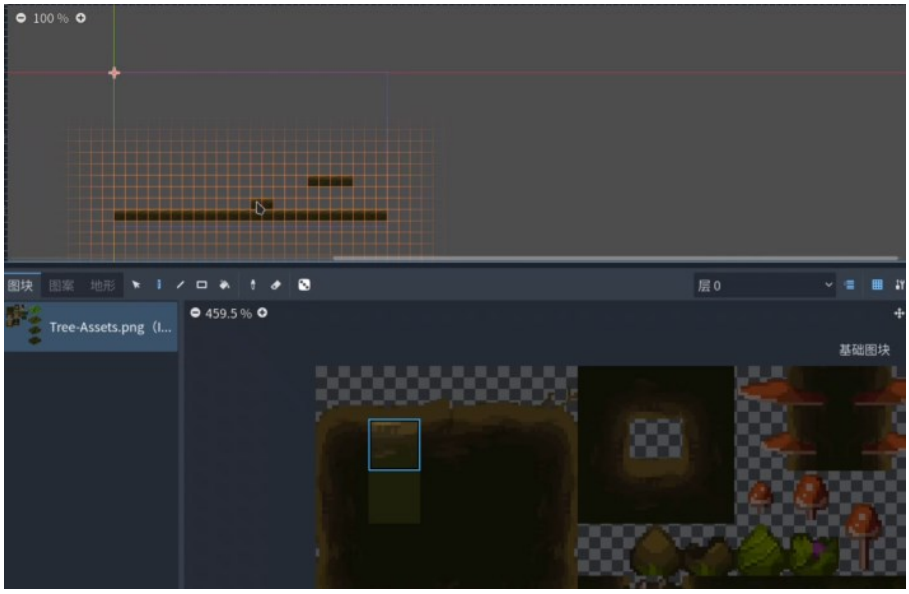

Figure 2. Scene module foundation construction (Photo/Picture credit : Original)

Add the player scene to the 2D camera and write some functions to adjust the following speed and amplitude of the picture angle. In module settings, check horizontal and vertical movement. .

If the player wishes to use the built-in module of the engine to enhance the TileMap section, they must first select the required area in the pixel grid and use the built-in function to quickly draw the map. Prior to this, the player needs to set the probability refresh of all modules to avoid generating too many repetitive maps in the same area. If the player needs to decorate the map, they need to set the top layer and use the engine's own random function to generate decorations. For a ParallaxBackground, create a new ParallaxBackground for both the inner and outer layers. This step includes setting up the background and creating covering vegetation in front of the character, as shown in figure 3.

Figure 3. Parallax background finish map (Photo/Picture credit : Original)

DOI: 10.54254/2755-2721/110/2024MELB0110

2.2. Design scrolling of the background

One characteristic of a fighting game is the modification of node attributes within the game to achieve the movement and infinite scrolling of the background. To accomplish this, the MIR attribute of the Parallax layer is initially utilized to enable infinite scrolling. Adding Parallax layer nodes and repeating background images achieves the movement of objects such as trees (Figure 4).

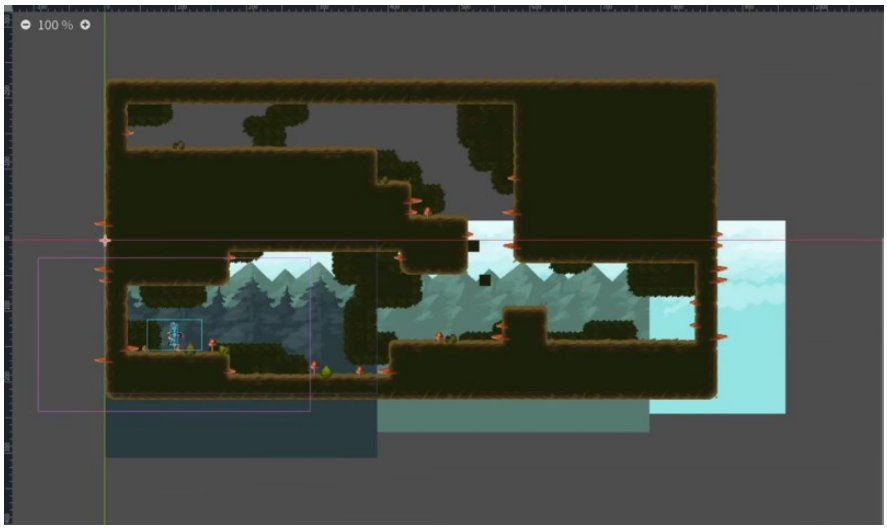

Figure 4. Parallax background makes part of the content (Photo/Picture credit : Original)

To set up player control over the character's movement, the first step is to adjust the character's movement speed and jump height. Next, incorporate acceleration and deceleration processes and refine character turning. Further, set up how the character accelerates while standing or in the air, including adding a countdown timer, buffering the jump button press, and controlling jump height. These adjustments will contribute to a better gaming experience.

To determine if the character is off the ground, compare the is_on_floor property before and after the character moves. Implementing a character jump requires a step: jump as soon as the character touches the ground by adding the jump_request_timer node and meeting the conditions. The player also needs to control the character's jump height, which is determined by the speed at which the jump key is released.

If the player wishes to implement control functions more easily, they can set up a Finite-state automaton (FSA). An FSA is a mathematical model representing finite states and transitions and actions between those states. Creating this model helps developers add character states more readily, such as the wall-slide mechanic.

2.3. Design a wall slide in the engine

To implement wall sliding, first create an animation named "Wall Sliding" in the animation player and add necessary keyframes to define the character's motion while sliding down the wall. By adjusting the X-axis scale value of the nodes, the player character can flip left or right upon contact with the wall, simulating the effect of sliding along the wall. To enhance the efficiency of animation production, animation editing plugins can be utilized to simplify the keyframe addition process, ensuring smooth and natural animations.

Next, create a state named "Wall Sliding" in the state machine and write the relevant logic code. In the wall sliding state, the character's falling speed should be lower than the normal falling speed, as the character experiences friction while sliding (Figure 5). Additionally, it is necessary to determine the sliding direction based on the normal of the contact surface between the character and the wall, ensuring that the character moves correctly along the wall without drifting away.

DOI: 10.54254/2755-2721/110/2024MELB0110

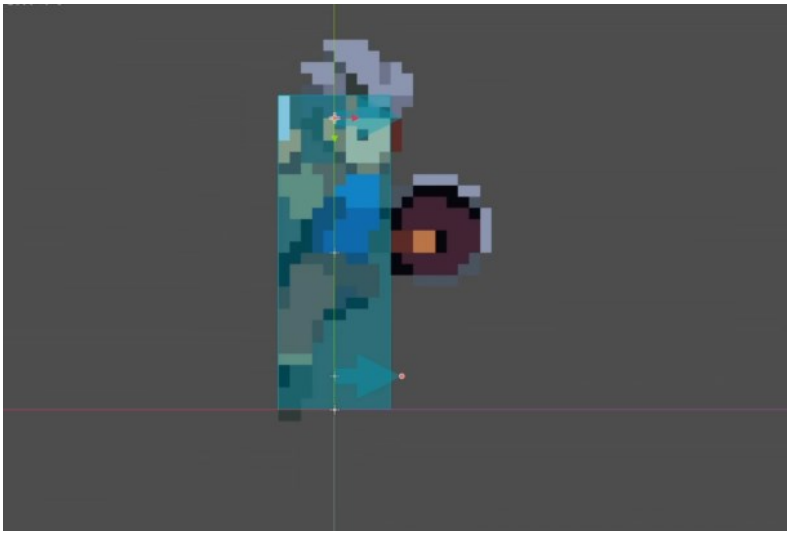

Figure 5. Slide wall key frame and direction direction (Photo/Picture credit : Original)

To implement wall jumping, first define a wall jump state in the state machine and set the initial speed parameters. In the state machine's "GetNextState" function, detect whether the player has pressed the jump key. If the jump key is detected, switch the state machine to the wall jump state. Next, in the state machine's "TransitionState" function, adjust the character's speed and direction to ensure that the character can correctly leave the wall during the wall jump and appropriately orient in the air. To enhance the visual experience, a slow-motion effect can be added at this stage, allowing players to see the details of each action clearly. Moreover, to make the wall jump more realistic, it is important to adjust the character's facing time before the wall jump and the acceleration during the jump. By setting these parameters appropriately, it is possible to avoid the character forming an S-shaped path during the wall jump, thereby enhancing the player's control experience. Finally, when the player jumps to the opposite wall, they should immediately enter the falling state rather than waiting for the character to land completely before proceeding to the next action, which will make the wall jump process more coherent (Figure 6).

Figure 6. Push off wall jumping process show (Photo/Picture credit : Original)

2.4. Design boar enemies and attack modules.

When creating the boar enemy, first define the enemy's scene structure, including necessary attributes such as health points, attack power, and movement speed, and add corresponding animations. The player needs to define the enemy's attributes and logic, including direction, speed, and acceleration parameters,

DOI: 10.54254/2755-2721/110/2024MELB0110

to ensure the enemy can act according to the game environment. Use images of the boar to create animations and add wall and cliff detection nodes to ensure that the enemy can change direction or stop moving when approaching boundaries. These detection nodes can prevent the enemy from passing through obstacles or falling off cliff edges.

Finally, utilize the concept of collision layers to create wall, cliff, and player detection nodes. By setting collision layers and collision masks, it is possible to accurately distinguish whether the collision object is a wall, a cliff, or the player. This will help the enemy in the game correctly identify and respond to different collision events.

In creating the attack module, the first step is to implement attack actions in the game, including adding animations, setting attack windows, and binding attack keys. For the first, second, and third attacks, three corresponding animation sequences should be added, with different cooldown times set between each attack action to ensure the continuity and realism of the attacks. In the attack animation, set a time window and require players to press the attack key within this timeframe. If the input system detects that the attack key has been pressed, it should enter the attack state and execute the entire attack logic. This includes playing the character's attack action animation, determining the attack hit, and calculating damage. Regarding the logic of the game's attack state transitions, it first involves judging the attack key, playing animations, and handling state transitions. Additionally, players must use variables to record attack conditions and determine the `is_combo_requested` flag to switch between different attack modes. For example, if the player presses the attack key consecutively, a combo attack can be triggered. Finally, all states should uniformly handle the physical logic of the three-tiered attack, ensuring that each attack correctly interacts with the enemy's collision volume and applies corresponding damage effects based on the attack type and power.

2.5. Design injuries and deaths.

First of all, in the game to realize the action of wild boar injury and death, and how to record and limit the blood amount of wild boar. Secondly, make injury and death animation, make the character disappear after being hurt, and add injury and death animation. And statistical information, and limit the value range of its blood volume. For example, make boar injured and dead status, make boar health decrease, and delete boar when health reaches 0. Then by modifying the properties to achieve the wild boar animation effect, injured state and death state switch, and record the damage information. Finally change the color and add keyframes to record the death (Figure 7).

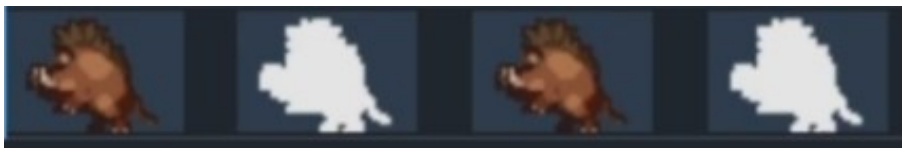

Figure 7. Injury and death animation (Photo/Picture credit : Original)

Create a status panel that includes player avatars and blood bars. Start by creating an empty scene and using the container node to horizontally row the child controls. Next, add avatar nodes, which players can use the Atlas texture to adjust the size and position to align with the avatar frame. Crafting Blood bars uses one of the two forms of progress bar that the system provides for fighting, which is made by adjusting the progress and border texture. Finally, write a function in the script to update the health by connecting a custom signal.

2.6. Design a slide tackle.

Need to make three stages of lying down, sliding shovel, and standing up and add corresponding animations. By adjusting the speed and adding condition judgment, a more realistic sliding effect can be achieved. For example, handling the timing of the sliding shovel action: Before entering the sliding shovel state, player can extend the animation playback time to handle the timing of the sliding shovel action. Create interactive objects in the game, including doors and stone tablets.

DOI: 10.54254/2755-2721/110/2024MELB0110

First, make interactive objects for scene switching and saving progress, then adjust collision layer and collision mask values, add key prompt animation and interaction logic, such as how to add interactive elements in the game, and solve the problem that interactive elements cannot respond to interaction. Then player need to add a rectangular collision area to the scene and add an INTERACTABLE node to make it interoperable. The interacting with variable was changed to an array and handlers were added to implement the capability of interacting with multiple objects simultaneously. For example, interacting with vegetables and mushrooms: By interacting with vegetables and mushrooms in the scene, the player's actions are realized. After that, the player has to deal with the death of the character in the game, and how to empty the interactive array so that it cannot interact. Handling Dead states: Determine the status in register interactive. If the current state of the state machine was dead, the interacting with clear method was called to empty the array when entering a dead state. Specific optimization can also be implemented such as having a boar hit and having the player knocked down, after which there is no interaction [8,9].

3. Result

This experiment collected the comments of people of different ages on this game (Table 1). The conclusion is given by comparison. Excellent Evaluation (15 points): The graphical design is flawless, with smooth character actions, excellent screen randomness, and operational feel, making it highly engaging and suitable for beginners. The game runs smoothly without any noticeable bugs, offering a fresh experience for first-time players. It is highly recommended and demonstrates strong competitiveness and originality in the market.

Moderate Evaluation (10 points): The graphical completion is high but requires improvements in screen resolution. There are minor detail issues, but the overall experience is acceptable. The game is better suited for players familiar with similar games, as beginners may find it challenging and less engaging. Minor bugs exist but do not significantly impact gameplay. The game is average and might be played during leisure time, though it is unlikely to be recommended to others. It has a high degree of similarity but offers interesting core gameplay and moderate competitiveness.

Poor Evaluation (5 points): The graphical design is generally acceptable but needs enhancements in UI and frame rate. The game scenes have significant defects and require substantial improvements. The content is dull and the gameplay is outdated, with numerous bugs that severely affect the user experience. The game quality is poor and not recommended. It shows signs of copying other games, lacks innovation, and has no market competitiveness.

Table 1. The test result

Tester	Appearance	Detail	Gameplay	Vulnerability feedback	First impression	Like comparison	Total points/90
Tester 1	15	15	15	10	10	10	75
Tester 2	15	15	10	15	10	15	80
Tester 3	10	15	15	15	10	15	80
Tester 4	10	15	10	15	10	10	70

According to the model, the lowest score was given to first impressions, with each of the four subjects scoring 10 points (Table 1). Details scored the highest, with all four participants scoring 15 points. This shows that the calculation model not only allows the data to be counted in six aspects, but also makes the three scores more user-friendly, which can help players get feedback from testers more easily. The final analysis also showed that games made by the Godot engine were more likely to have a high level of gameplay detail, but most players needed time to get used to the game's logic and graphics while playing.

DOI: 10.54254/2755-2721/110/2024MELB0110

4. Conclusion

This paper discusses how to use the godot engine to make a 2d horizontal fighting game, and uses the engine to show real cases and problems in the production process. The result is a relatively complete game. After that, the author analyzed the conclusions of different people on the playing process of this game by using real life experiments, and proved the advantages and disadvantages of godot engine. It is believed that Godot game Engine will gradually develop the community platform and plug-in modules in the future to meet the growing needs of game development. As the technology advances, we can expect more advanced graphics rendering and physics effects to be integrated into the engine. In addition, cross-platform support technology and the gradual maturity of virtual reality/augmented reality technology may also be one of the development directions. I believe that in the future, making a fighting game will become very easy, people can play their own games.

References

[1] Initial detection of inductance. (2019). *Amusement Equipment Engineering.*

[2] Wang, X. (2020). Preliminary exploration of play induction function *Master's thesis, University of Chinese Academy of Sciences.*

[3] Gregory, J. (2018). Game engine architecture. *A K Peters/CRC Press*.

[4] Hocking, J. (2018). Unity in action: Multiplatform game development in C#. *Manning Publications*.

[5] Bond, J. G. (2020). Introduction to game design, prototyping and development: From concept to playable game with Unity and C#. *Addison-Wesley Professional.*

[6] Anokolisa. (2024). Free - Pixel art asset pack - sidescroller fantasy - 16x16 forest sprites. *https://anokolisa.itch.io/sidescroller-pixelart-sprites-asset-pack-forest-16x16*

[7] Brullov. (2024). Generic character asset v 0.2. Itch.Io. https://brullov.itch.io/generic-char-asset

[8] Sonatina. (2024). Action RPG music pack: Infinity crisis COMPLETE. *Itch.Io. https://itch.io/embed/951119?linkback=true&link_color=5bddfa*

[9] Kenney. (2024). Assets. *https://www.kenney.nl/assets*