Global Journal of Engineering and Technology Advances

eISSN: 2582-5003

Cross Ref DOI: 10.30574/gjeta

Journal homepage: https://gjeta.com/

(REVIEW ARTICLE)

Game engine design and application based on C++

Depeng Xu [*], Yan Xie, Qing Lu and Mengshan Li

College of Physics and Electronic Information, Gannan Normal University, Ganzhou, Jiangxi 341000, China.

Global Journal of Engineering and Technology Advances, 2025, 22(02), 103-112

Publication history: Received on 02 January 2025; revised on 10 February 2025; accepted on 13 February 2025

Article DOI: https://doi.org/10.30574/gjeta.2025.22.2.0028

Abstract

With the rapid development of information technology, games have become an indispensable part of People's Daily life. The most important part of game software development is the design of game engine. This paper uses C++ high-level language to design and develop an efficient, stable and interactive game engine, using this game engine to develop a "hidden pairing" game software, game playability is high, this game engine has the characteristics of strong portability, robustness and scalability. All kinds of games can be designed and developed through game engine, which has a certain promotion effect on the development of game software industry

Keywords: Game Engine; C++; Hide Matchmaking Game Software; Hidden pairing

1. Introduction

With the development of science and technology, the development of the gaming industry has become increasingly important [1]. At present, most games on the market are developed using the Java language. However, in terms of controllability in games, the Java language is far inferior to C and C++ languages [2].

From the perspective of game design, the game engine is the heart of the game. If we compare a game to a car, the game engine is the engine of the car, directly determining its performance. The plot, levels, art, music, and gameplay that players experience in a game are all directly controlled and realized by the game engine. It plays the role of an engine, tying together all the elements in the game and directing them to work in an orderly manner. It ranges from controlling the volume level to calculating collisions, physical systems, and the relative positions of objects. The engine is the "main program used to control all game functions." Whether it is a 2D game or a 3D game, whether it is a role-playing game or an action shooter game, even the smallest game has such a piece of code that plays a controlling role[3-5].

2. Principles of Game Engine Design

All C++ programs under Windows start executing from the entry function WinMain(). The WinMain() function is mainly used for program initialization, displaying the main window, and entering the message loop to wait for and process messages. The operation mechanism of Windows applications is shown in Figure 1.

[*] Corresponding author: Depeng Xu.

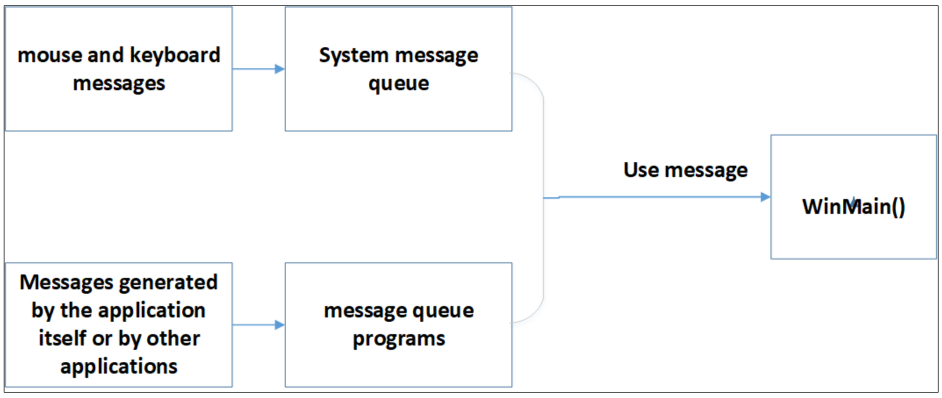
mouse and keyboard messages
System message queue
Use message
WinMain()
Messages generated by the application itself or by other applications
message queue programs

Figure 1 Running mechanism of windows applications

In the development of Windows applications, it is necessary to create program windows, and game programs are no exception. To establish a window, it is necessary to use window structures and related Windows API functions for creating windows, which provide an interface layer for communication between user applications and the operating system. When the program needs the operating system to perform a certain function, it is generally accomplished through API calls.

To create a window, one must first establish a window structure that contains all the relevant information about the window, such as menu styles. After the window structure is defined, the registration functions RegisterClass() or RegisterClassEx() are used to register it in the system. Once these preparatory steps are completed, the CreateWindow() function can be called to create the window. The window will not be displayed immediately after creation; it is also necessary to call the ShowWindow() function and the UpdateWindow() function to display the program window and update the client area of the window, respectively.

After the window is established, the program enters the message loop, waiting in the loop to receive and process messages. The typical message loop of a Windows application mainly involves three functions: GetMessage(), TranslateMessage(), and DispatchMessage(). The GetMessage() function retrieves messages from the thread's message queue, the TranslateMessage() function interprets the messages, converting virtual-key messages into character messages, and finally, the DispatchMessage() function sends the messages to the corresponding window procedure for further processing. In this process, the PeekMessage() function may also be used. The PeekMessage() function is similar to the GetMessage() function; it checks the thread message queue and, if a message exists, places it in the specified structure. In the program, message sending functions such as PostMessage() or SendMessage() can also be used to send specific messages.

After a message occurs in the program, the GetMessage() or PeekMessage() function will pass the message to the window, and the message handling function in the window structure will immediately process it.

An excellent game cannot be separated from a high-performance game engine. The game engine is the "main program used to control all game functions." That is, the game engine breaks down a game into multiple events, including engine initialization, parameter initialization, interaction, logic, rendering, resource release, and game over. The following figure shows the game engine architecture designed in this paper, as shown in Figure 2:

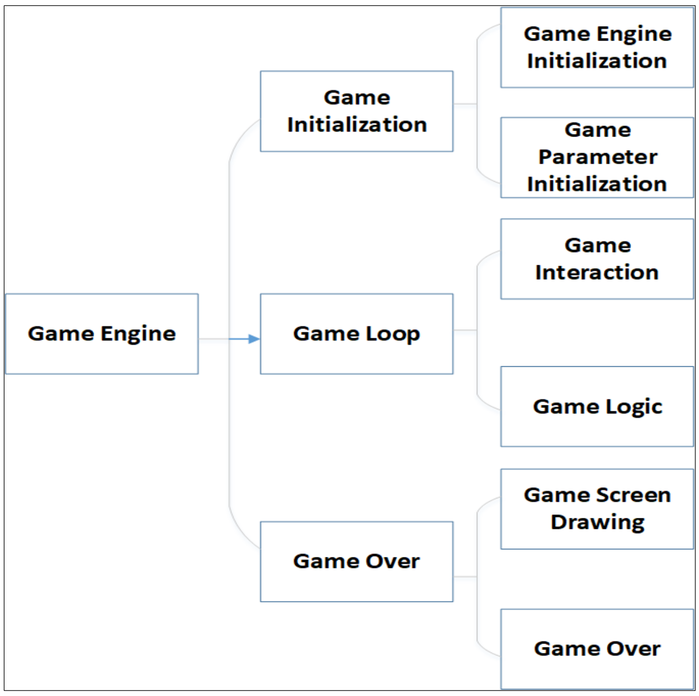

Figure 2 Game engine architecture

When the game starts running, the initialization event is responsible for loading the game's graphics and sounds, creating the game engine, that is, initializing the game engine, and creating the game window. The construction of the game engine mainly involves initializing engine parameters and constructing instances of the engine class, followed by the creation of the game window. After the window is successfully created, it becomes convenient to initialize the game parameters in the desired way. When the game is minimized or sent to the background and then restored, game pause and activation events will occur. When the game needs to draw itself, a game screen drawing event will occur. During the game, the player's control of the game mainly relies on the mouse and keyboard, which triggers game interaction events. Finally, before the game ends, the resources allocated in the program will be released.

During the operation of the game, the initialization process of the game can be divided into two parts: the construction of the game engine and the creation of the game window, both of which are encapsulated in the game engine and can be called by game developers in specific designs.

When program developers design game programs, game parameter initialization, game logic implementation, and game screen drawing, as functions unique to games, need to have these function methods clearly defined, providing a unified interface for program design. Functions such as the entry function WinMain() and the message processing function WindowProc(), which cannot be implemented in the game engine class, must be external functions. Due to the particularity of game programs, their WinMain() function is different from that of ordinary programs, which will be detailed in the following text [6-8].

Message processing in games is similar to that in general applications, all handled in the message processing function WndProc(). In game programs, the most basic message functions are game parameter initialization, screen drawing, game interaction, game pause, and game end with resource release. Game parameter initialization occurs after the game engine is constructed and the window is created. Since games vary, the specific functions need to be perfected by game developers according to design requirements [6]. Screen rendering is crucial for game programs, directly affecting players' gaming experience and interest. There are generally two ways of screen rendering: one is drawing in the game loop, and the other is drawing in response to the window drawing message WM_PAINT. During the operation of the game, the entire process is the interaction between the player and the game, which mainly relies on the mouse and keyboard, so the processing of mouse and keyboard messages is equally important. Sometimes when players need to leave the game window, the game execution needs to be paused and the game window suspended. Like general Windows programs, releasing the resources allocated in the program before the game ends is indispensable.

3. Game Engine Construction and Implementation

After designing the game engine in this paper, the Engine game engine is implemented through programming in the C++ advanced programming language.

3.1. Game Engine Construction and Parameterization

- The construction of the game engine mainly involves initializing engine parameters and building engine instances.

Engine (HINSTANCE hInstance, LPTSTR szWindowClass, LPTSTR szTitle, WORD wIcon, WORD wSmallIcon, BOOL bFullScreen, int nColorbit, int nWidth, int nHeight) {...}

- Creating the game window is mainly accomplished by the window creation function CreateWnd():
    - CreateGameWindow(){
    - wcAPP.cbSize = sizeof(wcAPP); // Assign values to window properties
    - wcAPP.lpszClassName = m_szWindowClass; // Set the window class name
    - wcAPP.style = CS_HREDRAW; // Define the window style
    - wcAPP.lpfnWndProc = WndProc; // Specify the message processing function
    - wcAPP.hInstance = m_hInstance;
    - wcAPP.cbWndExtra = 0;
    - wcAPP.cbClsExtra = 0;
    - /* Use the DEVMODE structure to set the screen display mode */
    - DEVMODE DevMode; // Clear the memory of the DevMode structure
    - ZeroMemory(&DevMode, sizeof(DevMode)); // Set the DevMode storage space to store screen property data
    - DevMode.dmSize = sizeof(DevMode); // Use the current screen display mode to fill the DevMode
    - EnumDisplaySettings(NULL, ENUM_CURRENT_SETTINGS, &DevMode);
    - BOOL bDisplayChange = FALSE; // Flag to indicate whether the screen display mode has changed
    - // Call the function CreateWindow to establish the window
    - hWnd = CreateWindow(m_szWindowClass, m_szTitle, dwWinStyle, nPosX, nPosY, nWndWidth, nWndHeight, NULL, NULL, m_hInstance, NULL);
    - ShowWindow(hWnd, SW_SHOWNORMAL);
    - UpdateWindow(hWnd);

Creating the game window requires setting the game display mode. If the game is in full-screen display mode, it is also necessary to set the display screen properties, set the screen size to the window size, and set the corresponding color mode. If it is not full-screen display, the window should be set to display in the center of the screen.

- Finally, use a GmInitialize() function to define content during the game, which facilitates subsequent development.

3.2. Game Engine Operation Mechanism

In the operation mechanism of the game engine, the main difference between its WinMain() function and that in ordinary programs lies in the processing of the message loop.

```
WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdline, int nCmdShow) {

  MSG msg;

  /* Initialize the game */

  if (!GmInitialize(hInstance)) return false;

  /* Message loop */

  if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)) {
        if (msg.message == WM_QUIT) // If it is a quit message, exit
            break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

First, the GmInitialize() function is called to create the game engine and initialize the game framework. Then, the game enters the message loop. By using PeekMessage() to receive messages, the game loop continues to execute when no messages occur, rather than waiting for messages. When messages occur, the game loop switches to process the messages; when no messages occur, the message loop is executed.

3.3. Game Engine Message Interaction

In the message interaction of the game engine, the message processing function WndProc() is used. The following introduces some commonly used messages in game programs:

```
WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    /* Use a switch statement to judge messages and perform corresponding processing */
    switch (message) {
    case WM_CREATE: // Window creation message
        Engine::GetEngine()->SetWindow(hWnd);
        GmStart(hWnd);
        break;
    case WM_PAINT: // Window drawing message
        HDC hDC;
        PAINTSTRUCT ps;
        hDC = BeginPaint(hWnd, &ps);
        GmPaint(hDC);
        EndPaint(hWnd, &ps);
        break;
    case WM_LBUTTONDOWN: // Left mouse button down message
        MouseLButtonDown(hWnd, LOWORD(lParam), HIWORD(lParam), wParam);
        break;
    case WM_SETFOCUS:
        GmActive(hWnd);
        Engine::GetEngine()->SetPause(FALSE);

        break;

    case WM_KEYDOWN: // Keyboard key press message

        if (wParam == VK_ESCAPE) // Check if the Esc key is pressed

            SendMessage(hWnd, WM_CLOSE, NULL, NULL);

        break;

    case WM_KILLFOCUS:

        GmPause(hWnd);

        Engine::GetEngine()->SetPause(TRUE);

        break;

    case WM_CLOSE:

        if (GameWindowClose(hWnd))

            DestroyWindow(hWnd);

        break;

    case WM_DESTROY:

        GmEnd();

        PostQuitMessage(0);

        break;

    default:

        return DefWindowProc(hWnd, message, wParam, lParam);

    }

}
```

4. Application of the Game Engine --- "Hidden Pair" Game Software

The game engine designed and developed in this paper is used to implement a "Hidden Pair" game software.

4.1. Design Scheme of the Hidden Pair Game Software

The Hidden Pair game is a casual puzzle game with elements of the traditional "spot-the-difference" series. It is simple to operate and easy to get started. The game initially combines elements of classic picture-finding series games and RPG game elements. Players cultivate characters, complete tasks, and upgrade various skills in the game. They can also use props to help overcome difficulties and find targets when facing challenges.

The Hidden Pair game is developed using the C++ advanced programming language, employing methods such as game engines, arrays, functions, and structures to create an intellectual game. The following figure shows the running window of the Hidden Pair game software, as shown in Figure 3.

Figure 3 The game software running window of hidden pairing

When the game starts, players search for items on the left that have similar outlines to the three black patterns on the right. The game begins with a timer starting and a set number of clicks allowed. When a correct match is found, the black item lights up, and simultaneously, one star lights up. Only by successfully finding all three patterns within the given number of clicks and time limit can the player pass the level. In the next level, both the time and the number of allowed clicks will decrease accordingly. At the beginning of the game, the player starts with 60 coins. If the player runs out of clicks during the game, they can click a button to increase the number of clicks by 2, at the cost of 20 coins. Once the coins are used up, no more clicks can be added.

4.2. Functional Modules of the Game Software

The design of the Hidden Pair game software includes five major functional modules: random image sorting, image correspondence, pause function, level function, and coin function. The structure of these functional modules is shown in Figure 4 [9-11].

4.2.1. Random Image Sorting Function

After the game starts, the left side of the scene is evenly divided into 18 positions, and the 18 item images are randomly sorted. When proceeding to the next level or restarting the current level, these 18 item images will be randomly sorted again. On the right side of the game scene, there are three black candidate patterns. These three black patterns match the outlines of three of the 18 item images on the left. Whenever the player moves to the next level or restarts the current level, these three black candidate patterns will also reappear randomly.

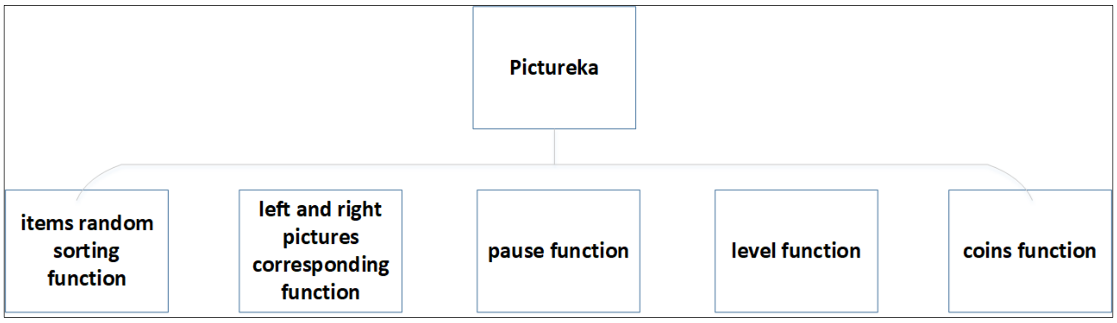
Pictureka
items random sorting function
left and right pictures corresponding function
pause function
level function
coins function

Figure 4 The game software function module structure of hidden pairing

4.2.2. Image Correspondence Function

When the player clicks on an item image on the left, the black candidate pattern on the right will only light up if the clicked item image has a similar outline to the black pattern. At the same time, a black star in the upper right corner of the scene will also light up accordingly. When moving to the next level or restarting the current level, even if all images are rearranged, the images with similar outlines on both sides can still correspond one-to-one.

4.3. Algorithm Implementation of the Game Software

```
/* Initialize patterns and positions */

// Initialize random number seed

srand((unsigned)time(NULL));

for(i=0; i<ROWS*COLS; i++){

  do{

    // Randomly generate a position number m among the 18 positions

    m = rand() % (ROWS*COLS);

  } while(array2[m] == 0); // Array array2 is used to mark whether the position is occupied

  // Mark the image sequence number corresponding to this position

  array1[m] = i;

  // Mark this position as occupied

  array2[m] = 0;

}

for(i=0; i<ROWS; i++){

  for(j=0; j<COLS; j++){

    // Array g_nCardPattern is used to represent the positions of the 18 images

    // Pass the value at this position in array array1 to array g_nCardPattern

    g_nCardPattern[i][j] = array1[i*6+j];

  }

}

// Candidate patterns

int arr[18];

for(i=0; i<3; i++){

  do {

    // Randomly generate a position number n

    n = rand() % 18;

  } while(arr[n] == 0);  // Check if the position is occupied

  // Mark the image sequence number corresponding to this position

  g_nCardarray[i] = n;
  // Mark this position as occupied

  arr[n] = 0;

}
```

The checkposition() function: // Check the position of the image corresponding to the candidate pattern

```
int checkposition(int num){

  int i, j;

  for(i=0; i<ROWS; i++){

    for(j=0; j<COLS; j++){

      /*Match the image sequence number at each position with the target image sequence number according to the position order*/

      if(g_nCardPattern[i][j] == num){

        return(i*6+j);

        break;

      }

    }

  }

  return 0;

}
```

The game is debugged and runs in the Windows system environment. The game interface is aesthetically pleasing, the operation is convenient, and the stability and robustness of the game are good. The game design adds new features such as coins and time, enhancing the challenge of the game, making it more playable and enjoyable for players

5. Conclusion

This paper designs and develops an Engine game engine based on C++. The designed game engine can be used to develop various types of games. Through the design and implementation of the Hidden Pair game software, it can be concluded that the game engine operates stably, with strong portability, good robustness, and high expandability. The code reuse rate of the game engine has been significantly improved, making the development and maintenance of games more convenient. This provides a certain reference for the development and design of other games.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Huang Wei, Liang Jilin, Zhai Song, et al. A Brief Discussion on C++ Game Writing [J]. Computer Enthusiast, 2014, 1(upper): 12.

[2] Guo Qi Yao. Design and Discussion of Double-Column Hanoi Tower in C Language Environment [J]. China Science and Technology Information, 2011(21): 64.

[3] Michael Morrison. Beginning Game Programming [M]. Sans Publishing, 2005:50.

[4] Michael Dawson. Beginning C++ Game Programming [M]. Course Technology PTR, 2004:89.

[5] Men Tao, Lun Shuxian, Ma Huan, Wang Xiaorong, et al. Implementation of C++ Game Engine [J]. Journal of Bohai University (Natural Science Edition), 2007, 28(3): 279-281.

[6] Li Peng, Jia Chuanjun, et al. C++ Game Programming [M]. Beijing: Tsinghua University Press, 2004:45.

[7] Zhai Jun Chang. A Brief Analysis of Game Engine Development [J]. Journal of Changchun Normal University: Natural Science Edition, 2006.25(1): 55-58.

[8] Wang Changjian. Development of 3D Game Engine Based on C++ and DirectX Technology [J]. Computer Knowledge and Technology, 2015, 11(27): 64-66.

[9] Shan Xueguang, Yang Qinghong, Jiao Li, et al. Application Research on the Non-Recursive Implementation Method of Recursive Functions [J]. Electro-Education Research, 2010(4): 72-74.

[10] Xiao Guiyun, Yuan Yali, et al. Methods and Process Analysis of Solving Hanoi Tower Problem with C Language [J]. Journal of Hebei North University: Natural Science Edition, 2002(3): 71-73.

[11] David H. Eberly. 3D Game Design: A Practical Approach to Real-Time Computer Graphics Second Edition [M]. Beijing: People's Posts and Telecommunications Publishing House, 2009:68.