Table of Contents

Treasure Hunter Game

INTRODUCTION

Treasure hunter is a single player game, where the player must collect a shovel, to uncover the buried treasure and bring it back to the ship. There are crab enemies that must also be avoided.

REPORT

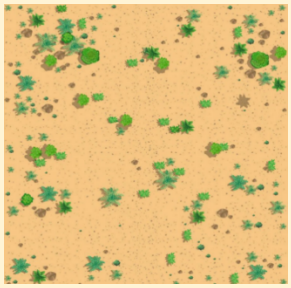
CONTROLS

Keyboard Key	Movement
W	Up
A	Left
S	Down
D	Right
Space	Dig
Esc	Close/Escape

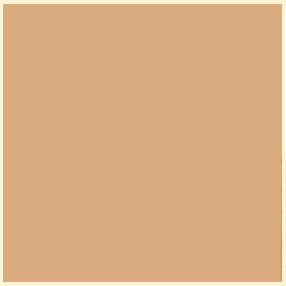
VISUAL STYLE AND THEMES

My final art style was dramatically different to what I had envisioned. I had ended up choosing very basic primitive models as I chose to focus on implementing the core gameplay. I would have liked my game to look like my original reference art, but I had struggled to create or find my own 3D models within the time frame.

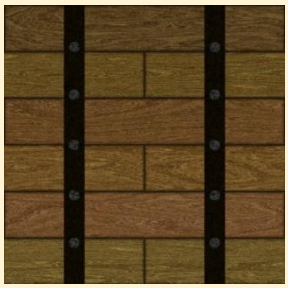
Textures Used



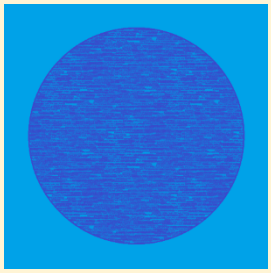
Ground



Wall



Treasure Chest & Gangway



Player

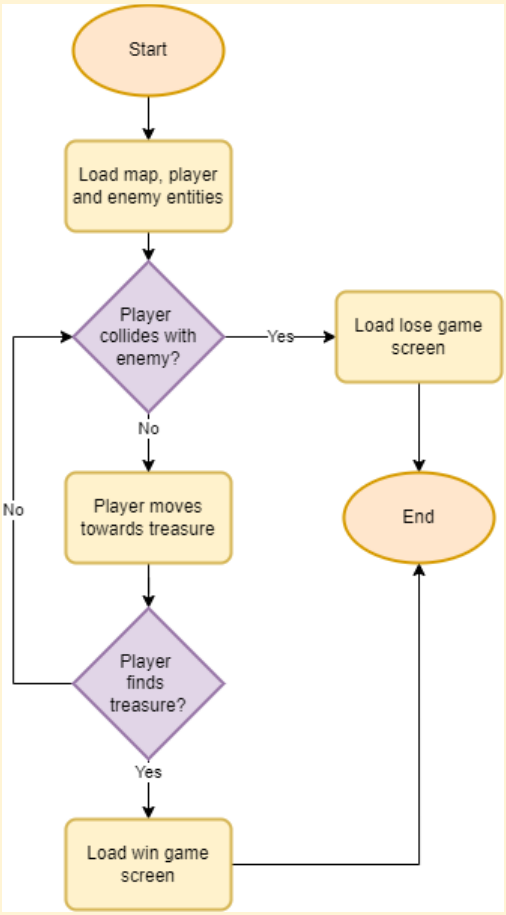
Crab

Gangway

Dig Spot

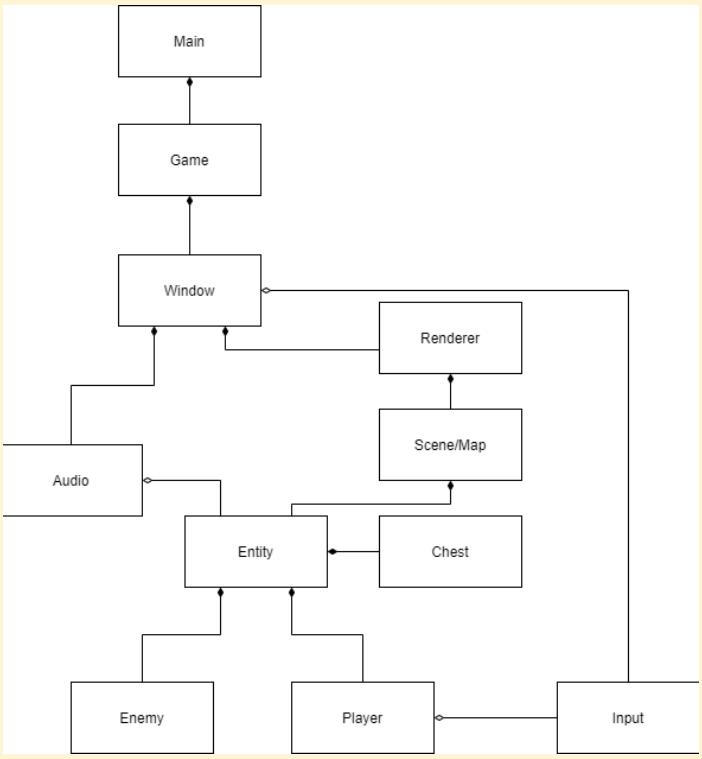
DESIGN DOCUMENTS

HIGHER-LEVEL GAME LOOP FLOWCHART

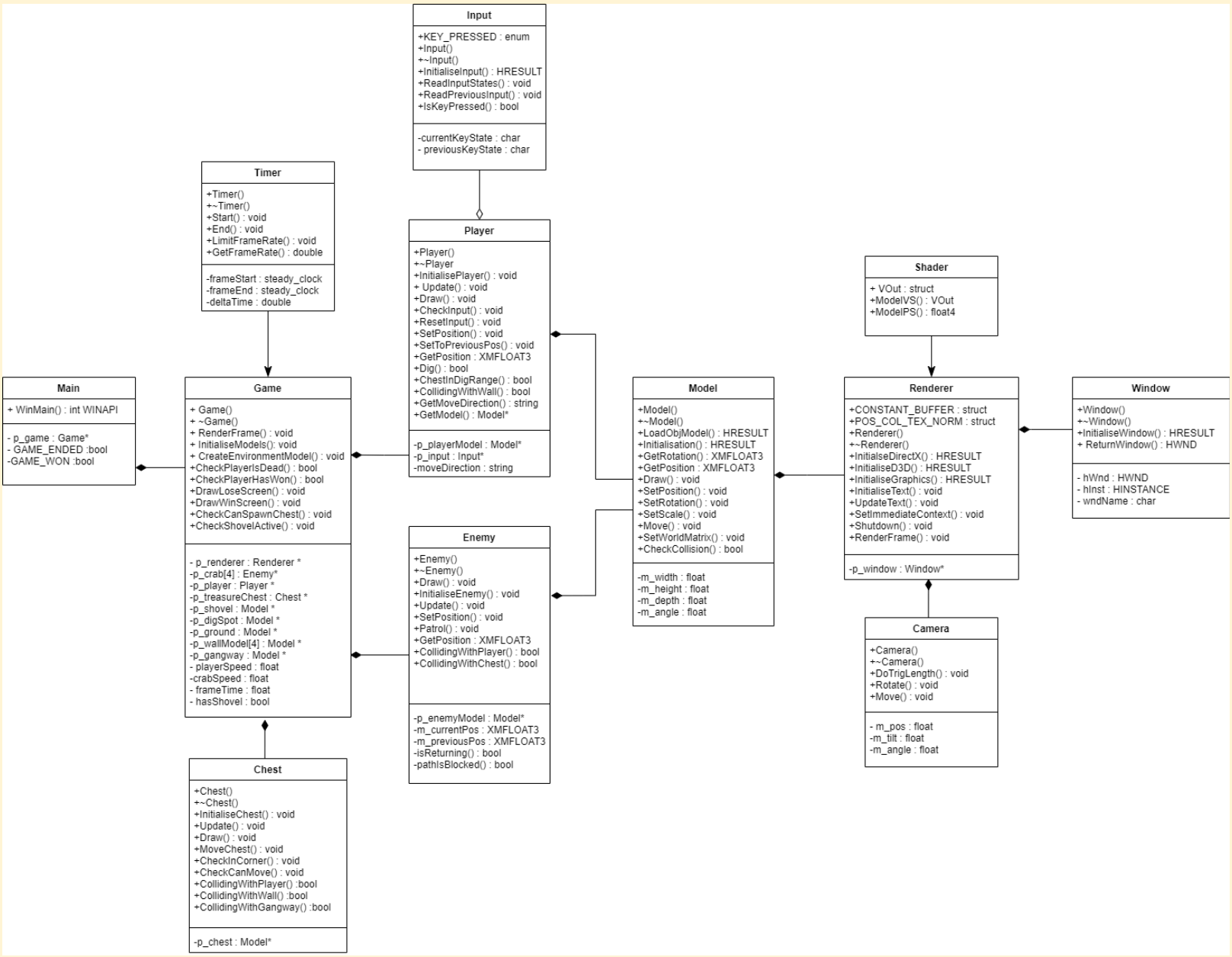


CLASS DIAGRAM

Prior Class Diagram



Final Class Diagram



TESTING

Test No.	Test	Description	Expected Result	Result Met
1	Game Initialises	Backend components of the game are initialised	-Window created -Renderer created -Viewports created -Buffers created -Shaders created -Basic input created	
2	Game Exit Condition	Make sure the player can quit the game using the escape key	When the escape key is pressed, the game exits	
3	Components Display	Components display within the window	Map and entities are shown on the screen. The map is at an isometric angle.	
4	Input	The program recognises when keys are pressed	Print key presses in the console window	
5	Player Movement	Attempting to move the player in all 4 directions	Player character moves in the corresponding directions when movement keys are pressed	
6	Player collision checks with the environment	Ensure the player collides with all elements of the environment correctly	The player does not walk/fall through any parts of the environment where they aren't supposed to	
7	Crab enemy movement	Enemy patrols up and down the expected path	The enemy doesn't change paths upon collision with other objects	
8	Collision checks with crab	The game detects collisions between the player and crab enemy	The collision is detected, and the player dies	
9	Limit frame fate	Game limits frame rate to target frame rate.	Frame rate is limited to 30 FPS	
10	Chest collision with player	The collisions between the player with the chest are detected properly	When the player collides with a chest it moves in the appropriate direction	
11	Chest collision with enemies	The collisions between the enemies with the chest are detected properly	When the enemies collide with the chest, they stop moving.	
12	Chest collision with gangway	The collision between the chest and the gangway are detected properly	When the chest collides with the gangway, a message is printed	
13	Player does not fall off edge	Player is not able to walk off the gap in the walls	Player treats gangway like it is a wall	
14	Game Loop works	When the player has either won or lost the game exits its loop correctly	If the game is won, the game win message is displayed. If the game is lost, the game lost message is displayed. Program is then closed.	
15	Font is displayed	Window displays font as a user interface overlay	Font is displayed in the form of a user interface.	

Test Number 1 – Game Initialises

The window is generated when the application opens, and the basic background is created.

Renderer.h shaders.hlsl

chestMovement (Global Scope)

```
1   #pragma once
2   //----------------------------------------
3   #define _XM_NO_INTRINSICS_
4   #define XM_NO_ALIGNMENT
5   //----------------------------------------
6   #include<dxgi.h>
7   #include<D3DX11.h>
8   #include<Windows.h>
9   #include<DxErr.h>
10  #include<DirectXMath.h>
11  #include<iostream>
12  //----------------------------------------
13  #include"SmartPointer.h"
14  #include"Window.h"
15  #include"Camera.h"
16  #include"text2D.h"
17  #include"Shader_Intigration.h"
18  //----------------------------------------
19  using namespace DirectX;
20  //----------------------------------------
21
22  struct CONSTANT_BUFFER0
23  {
24      XMMATRIX WorldViewProjection;    //64 bytes as it's 4 floats x 4 bytes
25      XMVECTOR DirectionalLightVector; //16 bytes
26      XMVECTOR DirectionalLightColour; //16 bytes
27      XMVECTOR AmbientLightColour;     //16 bytes
28      //Total size = 112 bytes therefore no packing bytes!!! yay!
29  };
```

shaders.hlsl

(Global Scope) texture0

```
Texture2D texture0;
SamplerState sampler0;

cbuffer CBuffer0
{
    matrix WVPMatrix;              //64 bytes
    //Lighting
    float4 DirectionalLightVector; //16 bytes
    float4 DirectionalLightColour; //16 bytes
    float4 AmbientLightColour;     //16 bytes
}

struct VOut
{
    float4 position : SV_POSITION;
    float4 color : COLOR;
    float2 texcoord : TEXCOORD;
};

VOut VShader(float4 position: POSITION, float4 color : COLOR, float2 texcoord : TEXCOORD, float3 normal: NORMAL)
{
    VOut output;
```

Window.h | shaders.hlsl

chestMovement (Global Scope)

```
1   #pragma once
2   //--------------------------------------
3   #include<d3d11.h>
4   //--------------------------------------
5   class Window
6   {
7   public:
8       Window();
9       ~Window();
10      HRESULT InitialiseWindow(HINSTANCE hInsance, int nCmdShow);
11      HWND ReturnWindow() { return hWnd; }
12
13  private:
14      HWND hWnd = NULL;
15      HINSTANCE hInst = NULL;
16
17      char WndName[100] = "Fate's Window\0";
18
19  };
20
21
```

Test Number 2 – Game Exit Condition

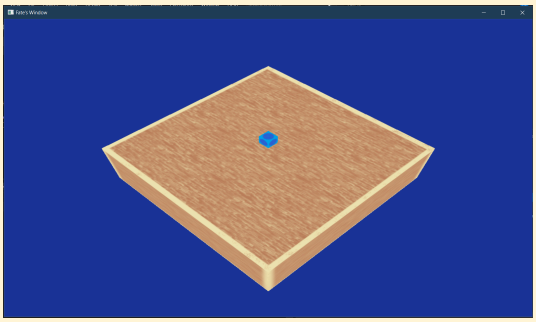
Game runs until the player quits the application.

```
while (msg.message != WM_QUIT)
{
    //PeekMessage() displays a message if it's relevant.
    //Once it's displayed it's removed from the event queue with PM_REMOVE
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
    {
        p_game->RenderFrame();
        p_game->ResetInput();
    }
}
//
return (int)msg.wParam;
```

```
The thread 0x1364 has exited with code 0 (0x0).
The thread 0x1c08 has exited with code 0 (0x0).
The thread 0x335c has exited with code 0 (0x0).
The thread 0x3c30 has exited with code 0 (0x0).
The thread 0x5a2c has exited with code 0 (0x0).
The thread 0x2074 has exited with code 0 (0x0).
The thread 0x347c has exited with code 0 (0x0).
The thread 0xf88 has exited with code 0 (0x0).
The thread 0x2488 has exited with code 0 (0x0).
The thread 0x1c30 has exited with code 0 (0x0).
The program '[33200] enemyCollisions.exe' has exited with code 0 (0x0).
```

Test Number 3 – Components Display

Ground model primitive and player model primitive are displayed in the isometric format.



```
private:
    float m_cameraPosX{ 23 }, m_cameraPosY{ 40 }, m_cameraPosZ{ 7 };
    float m_cameraTiltX{ -0.015 }, m_cameraTiltY{ -0.025 }, m_cameraTiltZ{ -0.03 };
    float m_cameraAngle{ -45 };
    XMVECTOR m_pos{ 0 }, m_lookAt{ 0 }, m_up{ 0 };
```

Test Number 4 – Input

I logged the current and previous key states, so that I could make the player move one space at a time, as I thought that this would make the game feel more, 'boxy' which is my intention with the design.

```
//-----------------------------
enum KEY_PRESSED
{
    KEY_UP = NULL,
    ESC = DIK_ESCAPE,
    W = DIK_W,
    A = DIK_A,
    S = DIK_S,
    D = DIK_D,
    SPACE = DIK_SPACE
};
```

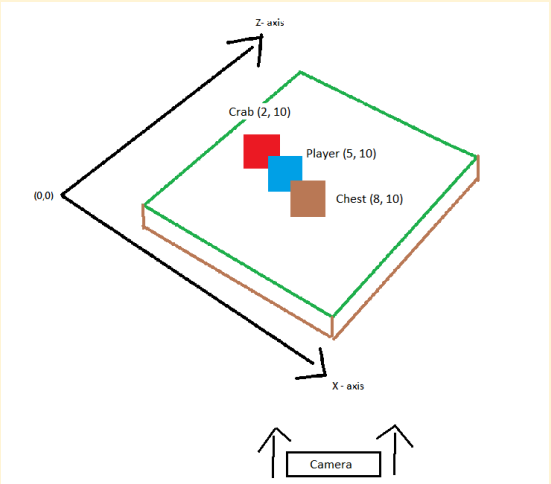
```cpp
void Input::ReadInputStates()
{
    HRESULT hr;

    hr = p_keyboardDevice->GetDeviceState(sizeof(currentKeyState), (LPVOID)&currentKeyState);

    if (FAILED(hr))
    {
        if ((hr = DIERR_INPUTLOST) || (hr == DIERR_NOTACQUIRED))
            p_keyboardDevice->Acquire();
    }
}

void Input::ReadPreviousInputStates()
{
    for (int i = 0; i < sizeof(previousKeyState); i++)
    {
        previousKeyState[i] = currentKeyState[i];
    }
}

bool Input::IsKeyPressed(KEY_PRESSED key) //DI_keycode  checks state of the key pressed
{
    return (currentKeyState[key] & 0x80) && !(previousKeyState[key] & 0x80);
}
```

```cpp
void Player::CheckInput(float speed, Model* wall1, Model* wall2, Model* wall3, Model* wall4)
{
    if (!PlayerCollidingWithWall(wall1, wall2, wall3, wall4))
    {
        //If player colliding with wall, they're returned to their previous position.
        m_previousPos = p_playerModel->GetPosition();

        if (p_input->IsKeyPressed(KEY_PRESSED::W))
        {
            p_playerModel->Move("x", -speed);
            OutputDebugString("W.\n ");
        }
        if (p_input->IsKeyPressed(KEY_PRESSED::A))
        {
            p_playerModel->Move("z", -speed);
            OutputDebugString("A.\n ");

        }
        if (p_input->IsKeyPressed(KEY_PRESSED::S))
        {
            p_playerModel->Move("x", speed);
            OutputDebugString("S.\n ");

        }
        if (p_input->IsKeyPressed(KEY_PRESSED::D))
        {
            p_playerModel->Move("z", speed);
            OutputDebugString("D.\n ");

        }
    }
    else p_playerModel->SetPosition(m_previousPos.x, m_previousPos.y, m_previousPos.z);
```

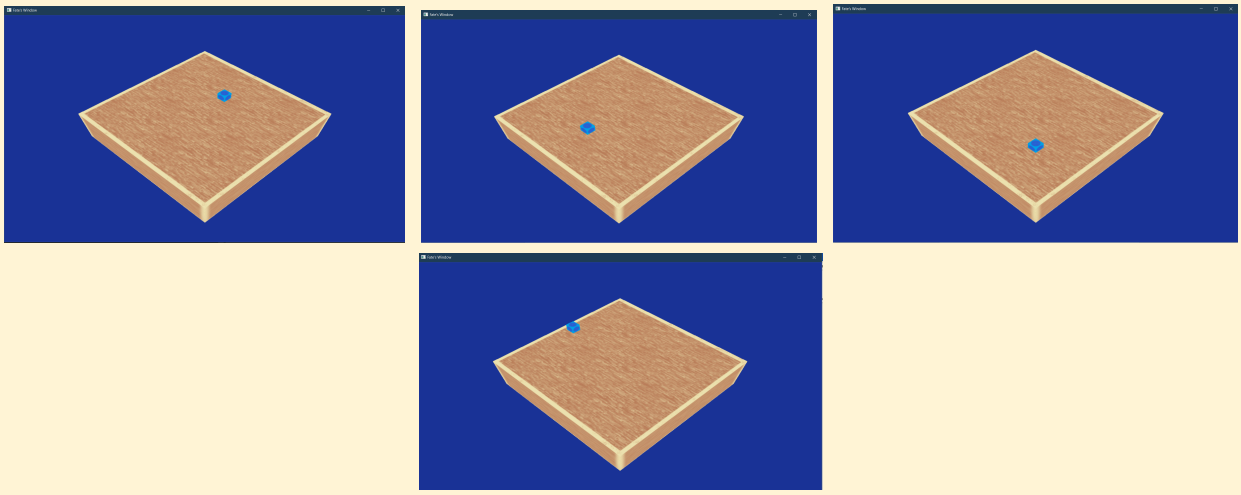
```
D.
W.
A.
S.
```

Test Number 5 – Player Movement

I had to swap the x and z-axis in relation to my initial design, due to my decision to rotate the camera negatively on the y axis.



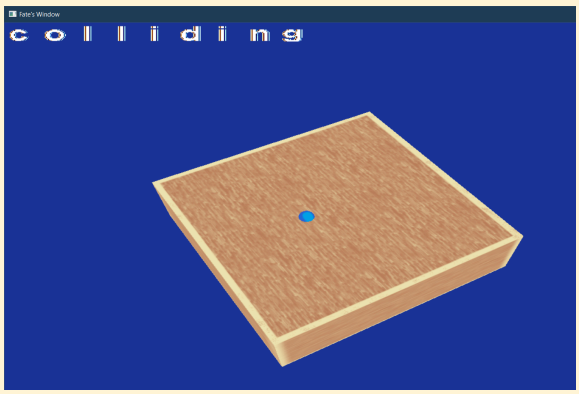

```
void Game::CheckInput()
{
    if (p_input->IsKeyPressed(KEY_PRESSED::W))
    {
        p_player->Move("x", -m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::A))
    {
        p_player->Move("z", -m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::S))
    {
        p_player->Move("x",  m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::D))
    {
        p_player->Move("z", m_speed);
    }
}
```

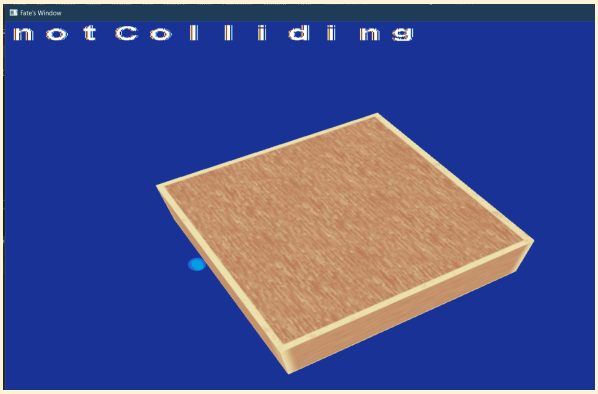


Test Number 6 – Player Collision Check with the Environment

```
bool Model::CheckCollision(Model* collidedModel)
{
    if ((m_x <= collidedModel->GetPosition().x + collidedModel->GetWidth())
            && (m_x + m_width >= collidedModel->GetPosition().x)
        && (m_y <= (collidedModel->GetPosition().y + collidedModel->GetHeight())
            && (m_y + m_height >= collidedModel->GetPosition().y
        && (m_z <= (collidedModel->GetPosition().z + collidedModel->GetDepth())
            && (m_z + m_depth <= collidedModel->GetPosition().z)))))
    {
        return true;
    }
    return false;
}
```

```
    p_renderer->ClearBuffer();
    ////p_renderer->UpdateText(aString);
    if (p_model->CheckCollision(p_ground))
    {
        //OutputDebugString(" colliding ");
        p_renderer->UpdateText("colliding");
    }
    else p_renderer->UpdateText("notColliding");
    //p_renderer->UpdateText(message);
```

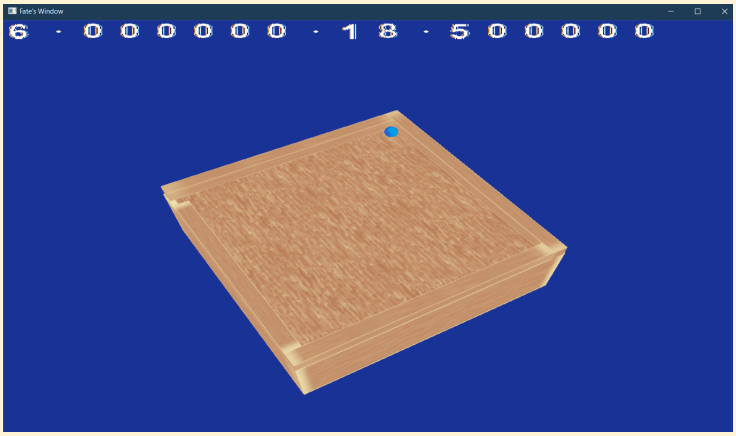




To stop the player from falling off the platform, I then added walls with colliders.

```
if (!PlayerCollidingWithWall())
{
    //If player colliding with wall, they're returned to their previous position.
    m_previousPos = p_player->GetPosition();

    if (p_input->IsKeyPressed(KEY_PRESSED::W))
    {
        p_player->Move("x", -m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::A))
    {
        p_player->Move("z", -m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::S))
    {
        p_player->Move("x", m_speed);
    }
    if (p_input->IsKeyPressed(KEY_PRESSED::D))
    {
        p_player->Move("z", m_speed);
    }
}
else p_player->SetPosition(m_previousPos.x, m_previousPos.y, m_previousPos.z);
```

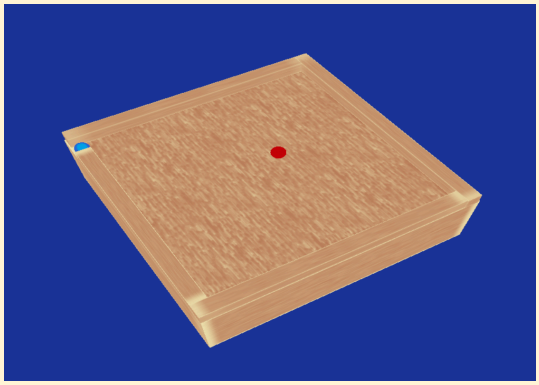
```
bool Game::PlayerCollidingWithWall()
{
    if (   p_player->CheckEnemyCollision(p_wallModel1)
        || p_player->CheckEnemyCollision(p_wallModel2)
        || p_player->CheckEnemyCollision(p_wallModel3)
        || p_player->CheckEnemyCollision(p_wallModel4))
    {
        return true;
    }
    return false;
}
```



Test Number 7 – Crab Enemy Movement

I had tried to implement the crab movement using my design documents (shown below), but this did not work.

```
Crab Movement

vector pos, start pos, end pos
bool returning, pathIsBlocked

Patrol (string direction)
    if (direction = z-axis and pathIsBlocked = false)
        if (pos.z != end pos.z and returning = false )
            pos.z + 1
        else if (pos.z = end pos.z and returning = false)
            return = true
        else if (pos.z != start pos and returning = true )
            pos.z - 1
        else if (pos.z = start pos and return = true)
            return = false
        end if
    else if (direction = x-axis and pathIsBlocked = false)
        if (pos.x != end pos.x and returning = false)
            pos.x + 1
        else if (pos.x = end pos.x and returning = false)
            return = true
        else if (pos.x != start pos and returning = true)
            pos.x - 1
        else if (pos.x = start pos and return = true)
            return = false
        end if
    else if (pathIsBlocked = true)
        remain at current pos
    end if
```



```
//y = depth, x = vertical, z = horizontal
void Model::Patrol(XMFLOAT3 startPos, XMFLOAT3 endPos, bool isHorizontal)
{
    if (!pathIsBlocked)
    {
        m_previousPos = m_currentPos;

        if (isHorizontal)
        {
            if (m_currentPos.z != endPos.z && !isReturning)
                m_currentPos.z += m_speed;

            else if (m_currentPos.z == endPos.z && !isReturning)
                isReturning = true;

            else if (m_currentPos.z != startPos.z && isReturning)
                m_currentPos.z -= m_speed;

            else if (m_currentPos.z == startPos.z && isReturning)
                isReturning = false;
        }
        else
        {
            if (m_currentPos.x != endPos.x && !isReturning)
                m_currentPos.x += m_speed;

            else if (m_currentPos.x == endPos.x && !isReturning)
                isReturning = true;

            else if (m_currentPos.x != startPos.x && isReturning)
                m_currentPos.x -= m_speed;

            else if (m_currentPos.x == startPos.x && isReturning)
                isReturning = false;
        }
    }
    else m_currentPos = m_previousPos;
```

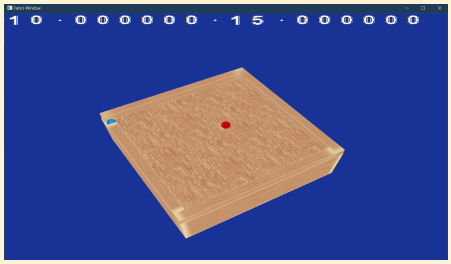
I found out that the crab was moving exactly as intended, but this result was not being drawn to the window.

```
std::string message = (std::to_string(m_currentPos.x) + "," + std::to_string(m_currentPos.z) + "\n");
OutputDebugString(message.c_str());
```

```
isHorizontal
0.000000,0.500000
isHorizontal
0.000000,1.000000
isHorizontal
0.000000,1.500000
isHorizontal
0.000000,2.000000
isHorizontal
0.000000,2.500000
```

```
0.000000,19.000000
isHorizontal
0.000000,19.500000
isHorizontal
0.000000,20.000000
isHorizontal
0.000000,20.000000
isHorizontal
0.000000,19.500000
isHorizontal
0.000000,19.000000
```

When I displayed the enemy position to the window, it showed that the enemy position was not being updated.



I had not updated the enemy's model position. So after I fixed that, the enemy moved.

```
if (!pathIsBlocked)
{
    m_previousPos = m_currentPos;

    if (isHorizontal)
    {
        if (m_currentPos.z != endPos.z && !isReturning)
            m_currentPos.z += m_speed;

        else if (m_currentPos.z == endPos.z && !isReturning)
            isReturning = true;

        else if (m_currentPos.z != startPos.z && isReturning)
            m_currentPos.z -= m_speed;

        else if (m_currentPos.z == startPos.z && isReturning)
            isReturning = false;

        m_z = m_currentPos.z;
    }
    else
    {
        if (m_currentPos.x != endPos.x && !isReturning)
            m_currentPos.x += m_speed;

        else if (m_currentPos.x == endPos.x && !isReturning)
            isReturning = true;

        else if (m_currentPos.x != startPos.x && isReturning)
            m_currentPos.x -= m_speed;

        else if (m_currentPos.x == startPos.x && isReturning)
            isReturning = false;

        m_x = m_currentPos.x;
    }
}
else m_currentPos = m_previousPos;
```

Test Number 8 – Crab Collision with Player

I implemented the collision inside the enemy class instead of the player class, as I thought that it would make more sense. As when I initialise an enemy it can take the singular player within its parameters, but if I had done it within the player, I would have to check for all enemy models in the scene.

```
void Enemy::Update(Player* player, XMFLOAT3 startPos, XMFLOAT3 endPos, bool isHorizontal, float speed)
{
    Patrol(startPos, endPos, isHorizontal, speed);
    if (CollidingWithPlayer(player))
    {
        OutputDebugString(" colliding\n ");
        playerIsDead = true;
    }
    //If colliding with chest, don't move/move back to last pos
}
```

```
bool Enemy::CollidingWithPlayer(Player* player)
{
    if (p_enemyModel->CheckCollision(player->GetModel()))
    {
        return true;
    }
    return false;
}
```

```
10.000000,12.999985
colliding
10.000000,12.929985
colliding
10.000000,12.859985
10.000000,12.789986
```

I then adjusted this to close the game if the player has died.

```
MSG msg = { 0 };
bool GAME_ENDED{ false };

while (msg.message != WM_QUIT && !GAME_ENDED)
{
    //PeekMessage() displays a message if it's relevant.
    //Once it's displayed it's removed from the event queue with PM_REMOVE
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
    {
        p_game->RenderFrame();
        if (p_game->CheckPlayerIsDead()) GAME_ENDED = true;
    }
}
//
return (int)msg.wParam;
}
```

```
10.000000,15.589973
10.000000,15.519974
Player Died.
 The thread 0x3f50 has exited with code 68 (0x44).
The thread 0xb1c has exited with code 68 (0x44).
The thread 0x788 has exited with code 68 (0x44).
The thread 0x298c has exited with code 68 (0x44).
The thread 0x4758 has exited with code 68 (0x44).
The thread 0x4a2c has exited with code 68 (0x44).
The thread 0x4be8 has exited with code 68 (0x44).
The thread 0x2e48 has exited with code 68 (0x44).
The thread 0x4cb8 has exited with code 68 (0x44).
The program '[8712] enemyCollisions.exe' has exited with code 68 (0x44).
```

```
bool Game::CheckPlayerIsDead()
{
    if (p_crab->CollidingWithPlayer(p_player))
    {
        OutputDebugString("Player Died.\n ");
        return true;
    }
    else return false;
}
```

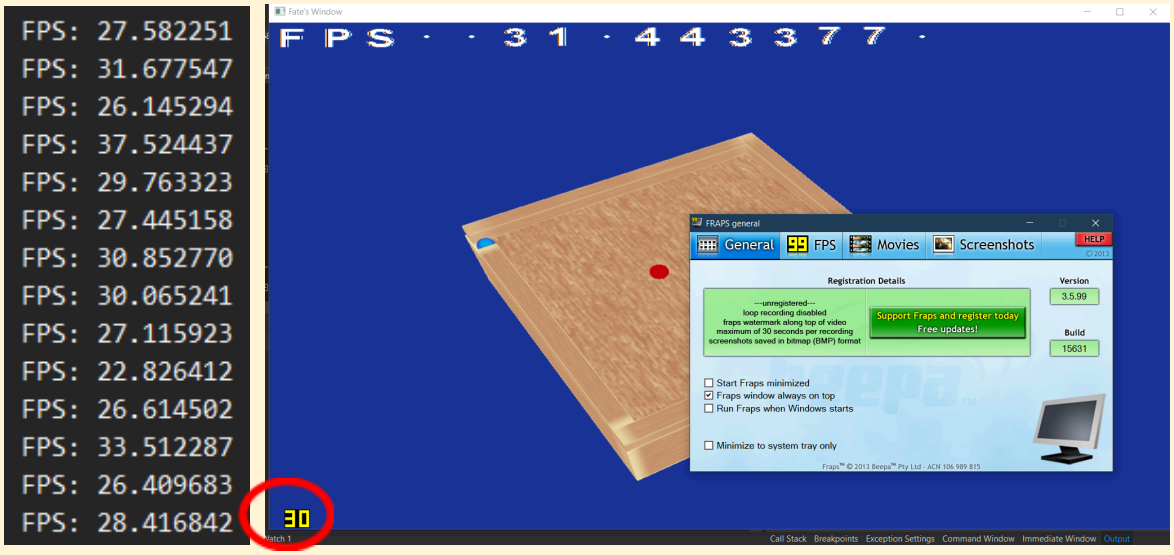
```
bool Enemy::CollidingWithPlayer(Player* player)
{
    if (p_enemyModel->CheckCollision(player->GetModel()))
    {
        return true;
    }
    return false;
}
```

Test Number 9 – Limiting Frame Rate

I decided to regulate the frame rate through the use of the 'chrono' time library, and the 'thread' library. I had decided to use the these with 'steady_clock', as this calculates the time more precisely than if I were to monitor time another way. I found that the websites gamesdev.net (Shaarigan 2022) and cplusplus.com (cplusplus 2022) were useful.

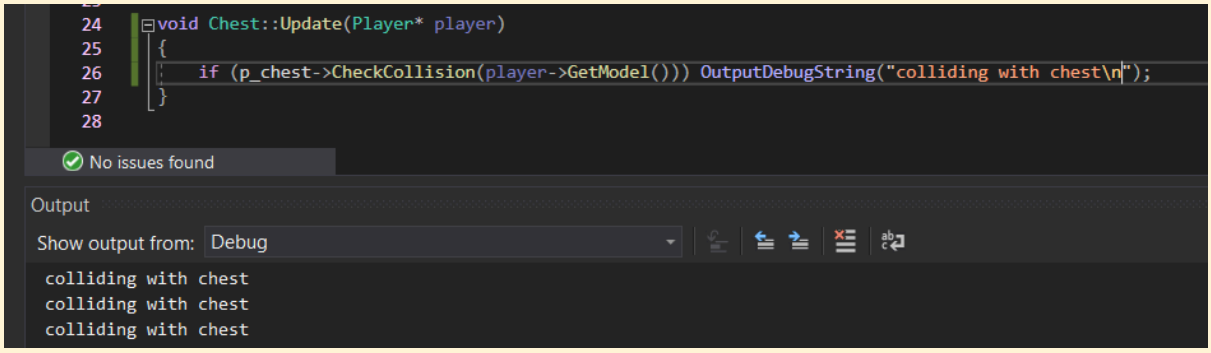
```
double Timer::GetFrameRate()
{
    chrono::time_point<chrono::steady_clock> currentFrame = chrono::steady_clock::now();

    auto elapsed = chrono::duration<double, milli>(currentFrame - frameStart);
    //1000 converts frame time to frame rate
    return 1000/elapsed.count();
}

void Timer::LimitFrameRate()
{
    //Casts the difference in time to milliseconds
    chrono::milliseconds sleepTime =
        chrono::duration_cast<chrono::milliseconds>(targetFPS - deltaTime);

    //Limits frame rate
    if (deltaTime.count() < targetFPS.count())
        this_thread::sleep_for(sleepTime);
}
```

I output the framerate to my game and output window. It proved to give quite consistent frame rate of 30FPS. I also used the app named Fraps (Beepa Pty Ltd 2022) to prove this.



Test Number 10 – Chest Collision with Player

I first checked that the chest registered player collision.



Next, I created the pseudocode logic for the chest movement.

```
previousPosition, currentPosition

Chest movement method

    //If player pushes the chest in space or player pushes the chest along the wall
    if ((collidingWithPlayer and not collidingWithWall) or
        (collidingWithPlayer and collidingWithWall and playerCollidingWithWall))

        currentPosition = previousPosition

        if(player moves up)
            chest moves up
        if(player moves down)
            chest moves down
        if(player moves left)
            chest moves left
        if(player moves right)
            chest moves right

    end if

    //If chest is pushed into the wall
    if(not collidingWithPlayer and collidingWithWall)
        currentPosition = previousPosition
    end if

    //If stuck on corner
    if(colliding with two walls)
        reset position to start position
    end if
```

I then added this design to my code and found that the player was able to push the chest into the walls.

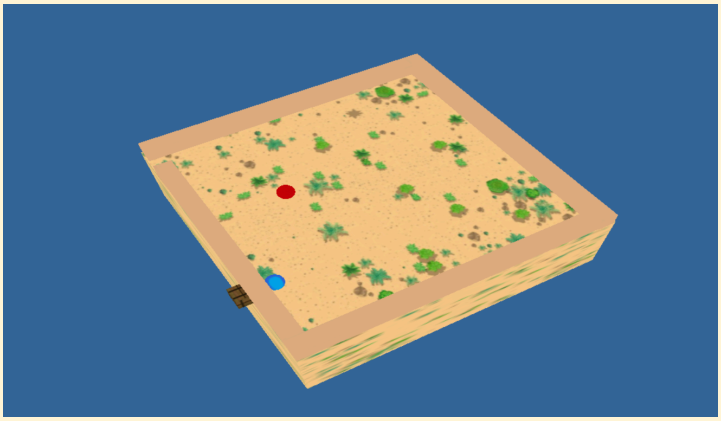
```
47
48  void Chest::MoveChest(Player* player, float playerSpeed)
49  {
50      /*_____________________________________________________
51      Used in CheckCanMove().
52      _____________________________________________________*/
53      string MOVE_DIRECTION = player->GetMoveDirection();
54
55      if (MOVE_DIRECTION == "up")
56      {
57          p_chest->Move("x", -playerSpeed);
58          OutputDebugString("Chest moved up\n");
59      }
60      else if (MOVE_DIRECTION == "down")
61      {
62          p_chest->Move("x", playerSpeed);
63          OutputDebugString("Chest moved down\n");
64      }
65      else if (MOVE_DIRECTION == "left")
66      {
67          p_chest->Move("z", -playerSpeed);
68          OutputDebugString("Chest moved left\n");
69      }
70      else if (MOVE_DIRECTION == "right")
71      {
72          p_chest->Move("z", playerSpeed);
73          OutputDebugString("Chest moved right\n");
74      }
75  }
```

0 2

Output

Show output from: Debug

```
E:\Computer Games Programming\DirectX\Scripts\ChestMovement\Model_S
Chest moved left
Chest moved left
Chest moved left
```



Right: chest stuck in the wall.

I then adjusted my code so that when the chest is colliding with the wall it resets the player's and chest's position to their previous position.

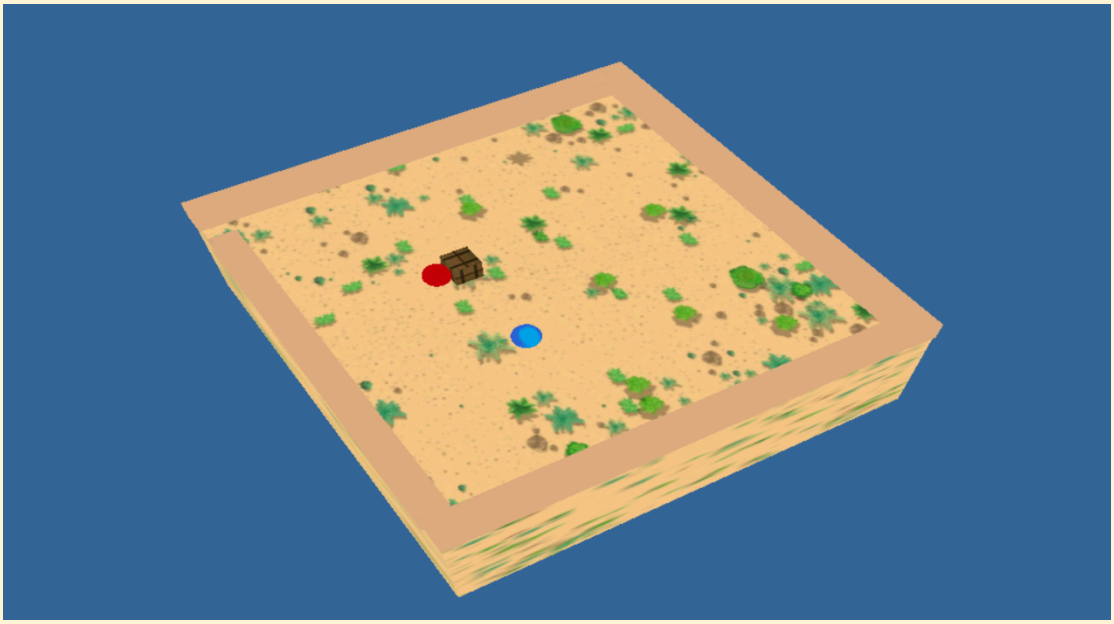
```
void Chest::Update(Player* player, Model* wall1, Model* wall2, Model* wall3, Model* wall4, float playerSpeed, XMFLOAT3 startPos)
{
    m_previousPos = m_currentPos;
    m_currentPos = p_chest->GetPosition();

    CheckCanMove(player, wall1, wall2, wall3, wall4, playerSpeed);

    if (CollidingWithWall(wall1, wall2, wall3, wall4))
    {
        player->SetToPreviousPos();
        p_chest->SetPosition(m_previousPos.x, m_previousPos.y, m_previousPos.z);
    }

    CheckInCorner(wall1, wall2, wall3, wall4, startPos);
}
```

And changed the corner check to check for the chest next position.

```
void Chest::CheckInCorner(Model* wall1, Model* wall2, Model* wall3, Model* wall4, XMFLOAT3 startPos)
{
    //XMFLOAT3 rightTopCornerPos{ 6,20,18 }, rightBottomCornerPos{ 15, 20, 19 }, leftBottomCornerPos{ 15, 20, 9.5 };
    /*_____________________________________________________________________________
    If chest is stuck in the corner.
    _____________________________________________________________________________*/
    if (p_chest->CheckNextPosition(wall1) && p_chest->CheckNextPosition(wall2) ||
        p_chest->CheckNextPosition(wall1) && p_chest->CheckNextPosition(wall3) ||
        p_chest->CheckNextPosition(wall1) && p_chest->CheckNextPosition(wall4) ||
        p_chest->CheckNextPosition(wall2) && p_chest->CheckNextPosition(wall3) ||
        p_chest->CheckNextPosition(wall2) && p_chest->CheckNextPosition(wall4) ||
        p_chest->CheckNextPosition(wall3) && p_chest->CheckNextPosition(wall4))
    {
        p_chest->SetPosition(startPos.x, startPos.y, startPos.z);
        OutputDebugString("Reset chest postion");
    }
}
```

```
Chest moved left
Chest moved left
Chest moved left
Reset chest postionThe
```

Test Number 11 – Chest Collision with Enemies



```
void Enemy::Update(Player* player, Chest* chest, XMFLOAT3 startPos, XMFLOAT3 endPos, bool isHorizontal, float speed)
{
    Patrol(startPos, endPos, isHorizontal, speed);

    /*
    If colliding with chest, don't move.
    */
    if (CollidingWithChest(chest)) pathIsBlocked = true;
    else pathIsBlocked = false;
}
```

Test Number 12 – Chest Collides with Gangway

Treasure chest successfully collides with the gangway and prints a message.

```
167
168 bool Game::CheckPlayerHasWon()
169 {
170     if (p_treasureChest->CollidingWithGangway(p_gangway))
171     {
172         OutputDebugString("Player has Won.\n ");
173         return true;
174     }
175     return false;
176 }
```

No issues found

Output

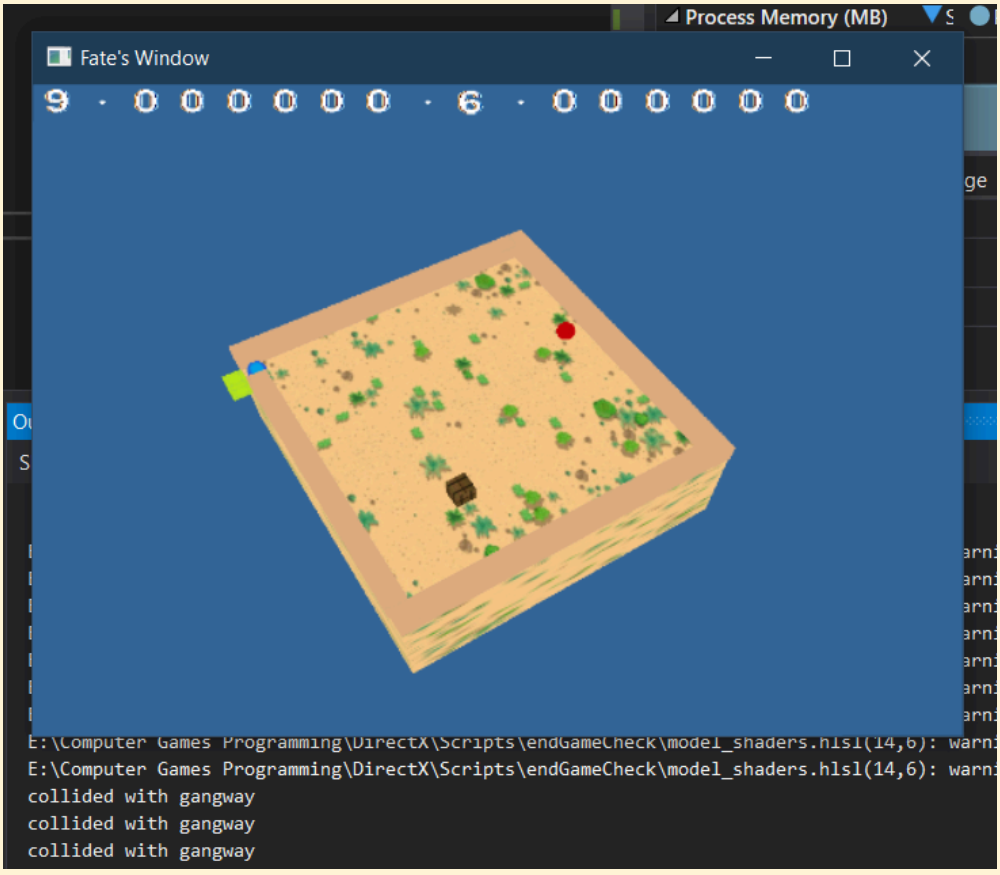
Show output from: Debug

Player has Won.

Test Number 13 – Player Does Not Fall Off Edge

When the player collides with the gangway, they are unable to fall off.

```
bool Player::PlayerCollidingWithWall(Model* wall1, Model* wall2, Model* wall3, Model* wall4, Model* gangway)
{
    if (p_playerModel->CheckCollision(wall1)
        || p_playerModel->CheckCollision(wall2)
        || p_playerModel->CheckCollision(wall3)
        || p_playerModel->CheckCollision(wall4)
        || p_playerModel->CheckCollision(gangway))
    {
        if (p_playerModel->CheckCollision(gangway))
        OutputDebugString("collided with gangway");

        return true;
    }
    return false;
}
```

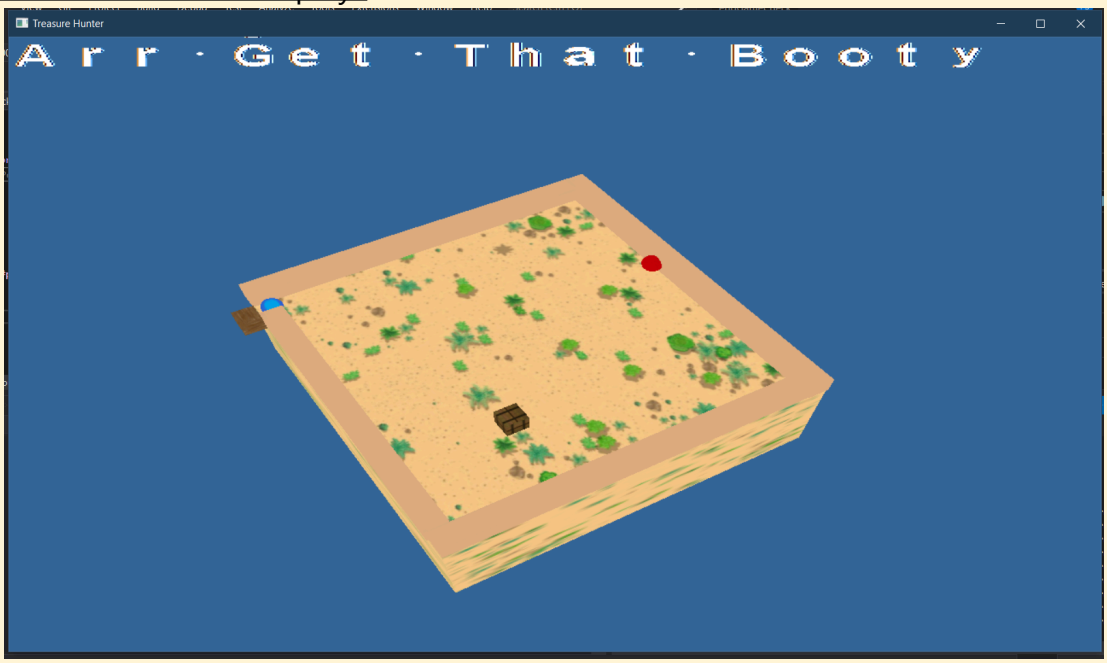


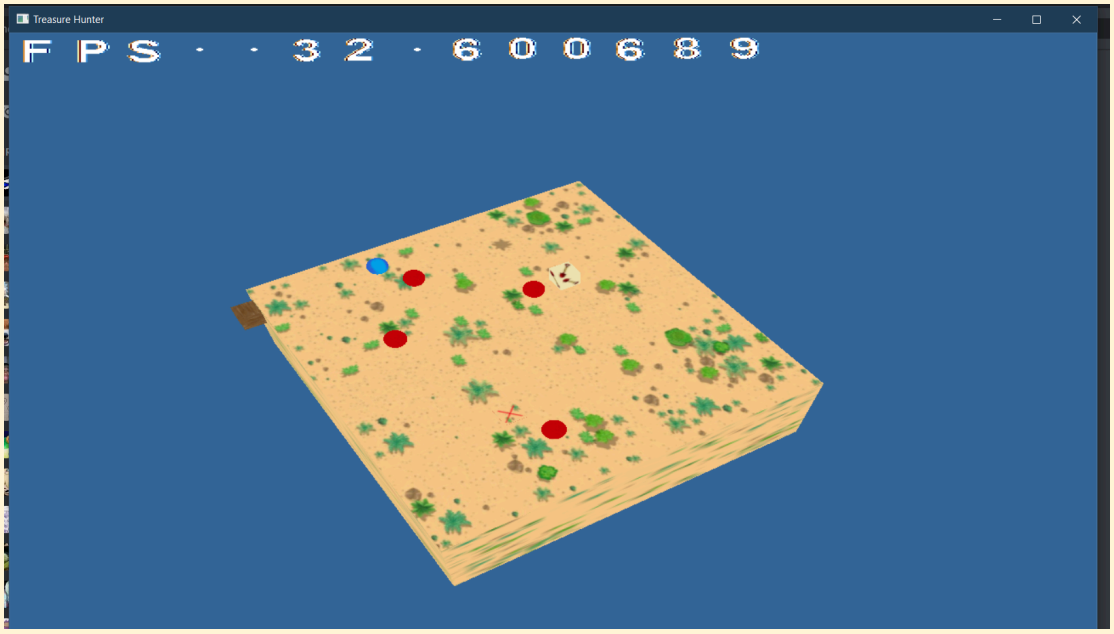
Test Number 14 – Game Loop Works

When the player collides with an enemy the end screen is displayed, and when the player pushes the treasure chest to the gangway, the win screen is displayed.

```
while (msg.message != WM_QUIT && !GAME_ENDED && !GAME_WON)
{
    //PeekMessage() displays a message if it's relevant.
    //Once it's displayed it's removed from the event queue with PM_REMOVE
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else //Run Game Loop
    {
        p_game->RenderFrame();
        if (p_game->CheckPlayerIsDead()) GAME_ENDED = true;
        if (p_game->CheckPlayerHasWon()) GAME_WON = true;
    }
}
if (GAME_ENDED)
{
    p_game->DrawLoseScreen();
    //Time for player to read screen.
    this_thread::sleep_for(chrono::seconds(3));
}
else if (GAME_WON)
{
    p_game->DrawWinScreen();
    //Time for player to read screen.
    this_thread::sleep_for(chrono::seconds(3));
}
//_____
return (int)msg.wParam;
```

Test Number 15 – Text Displays





Test Number 16 – Chest Spawns

The chest spawned when the player hit the dig key.

```
69
70  void Player::Dig()
71  {
72      //if player has shovel
73      if (p_input->IsKeyPressed(KEY_PRESSED::SPACE))
74      {
75          OutputDebugString("Digging\n");
76      }
77  }
78
```

0 1

Output

Show output from: Debug

Digging

However, the enemy did not move until the chest is found.

```
54      p_player->Update(m_PlayerSpeed, p_wallModel1, p_wallModel2, p_wallMo
55      if (p_player->Dig())
56      {
57          //If there's currently no chest in play, spawn a chest.
58          if (p_treasureChest == nullptr)
59          {
60              p_treasureChest = new Chest(
61                  p_renderer->GetDevice(),
62                  p_renderer->GetImmidiateContext(),
63                  (char*)m_chestTexture);
64
65              p_treasureChest->InitialiseChest(p_renderer->GetImmidiateCon
66              OutputDebugString("Chest spawned\n");
67  #ifdef p_treasureChest == null
68              OutputDebugString("Error: Treasure Chest uninitialised.\n")
69  #endif
70          }
```

No issues found

Output

Show output from: Debug

E:\Computer Games Programming\DirectX\Scripts\chestSpawning\model_shaders.hlsl(14,6): warni

E:\Computer Games Programming\DirectX\Scripts\chestSpawning\model_shaders.hlsl(14,6): warni

Chest spawned

I had realised that the method that checked the chest collision, did not return false if the chest was not active.

```
bool Enemy::CollidingWithChest(Chest* chest)
{
    if (chest != nullptr)
    {
        if (p_enemyModel->CheckCollision(chest->GetModel()))
        {
            return true;
        }
        return false;
    }
}
```

Therefore, I moved the 'return false' to outside the null pointer check and the enemy was able to move again.

Next, I added a check to see if the player was within range of the chest, in order for them to dig, and this worked successfully.

```
bool Player::ChestInDigRange(XMFLOAT3 chestPos)
{
    if (   p_playerModel->GetPosition().x - (p_playerModel->GetWidth()  * 2) <= (chestPos.x + 0.1f)
        && p_playerModel->GetPosition().x + (p_playerModel->GetWidth()  * 2) >= (chestPos.x - 0.1f)
        && p_playerModel->GetPosition().z - (p_playerModel->GetDepth()  * 2) <= (chestPos.z + 0.1f)
        && p_playerModel->GetPosition().z + (p_playerModel->GetDepth()  * 2) >= (chestPos.z - 0.1f))
    {
        return true;
    }
    return false;
}
```

Test Number 17 – Dig Spot is shown with Model

```
void Game::CheckCanSpawnChest()
{
    //If there's currently no chest in play, spawn a chest.
    if (p_treasureChest == nullptr && p_player->Dig())
    {
        //If player is next to chest spawn position
        if (p_player->ChestInDigRange(m_chestStartPos))
        {
            if (p_digSpot) delete p_digSpot; p_digSpot = nullptr;

            p_treasureChest = new Chest(
                p_renderer->GetDevice(),
                p_renderer->GetImmidiateContext(),
                (char*)m_chestTexture);

            p_treasureChest->InitialiseChest(p_renderer->GetImmidiateContext(), p_renderer->GetDevice(), (char*)m_cubeModel, m_chestStartPos);

#ifdef p_treasureChest == null
            OutputDebugString("Error: Treasure Chest not spawned.\n")
#endif
```

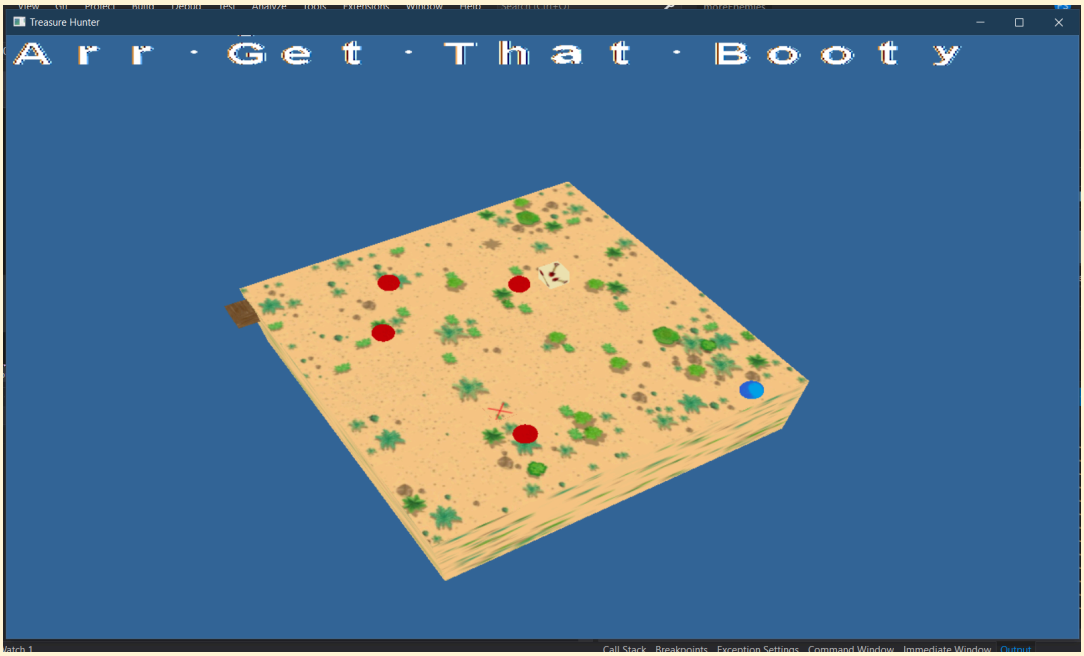
Below: model shows ‘x’ that marks where the treasure is.

Below: ‘x’ has disappeared when the treasure is uncovered.

Test Number 18 – Player requires chest to dig

```
void Game::CheckShovelActive()
{
    if (p_shovel != nullptr)
    {
        if (p_player->GetModel()->CheckCollision(p_shovel))
        {
            hasShovel = true;
            delete p_shovel; p_shovel = nullptr;
        }
    }
}
```

I added more enemies and stopped drawing the walls as my original design did not include walls, and makes the map look more like a floating island. So, this is what the final game looked like.



Additional Code

Smart Pointers

I have created my own smart pointers in my project and have successfully avoided memory leakage. I found the website geeksforgeeks.com (GeeksforGeeks 2021) helpful during implementation.

SmartPointer.h

chestMovement (Global Scope)

```
1   #pragma once
2
3   template<class T>
4   class SmartPtr
5   {
6   public:
7       explicit SmartPtr(T* p = nullptr) { ptr = p; }
8       ~SmartPtr() { delete (ptr); }
9
10      //Overloading dereferencing opereator
11      T& operator* () { return *ptr; }
12
13      //Overloading arrow operator so T can be accessed like a pointer
14      T* operator->() { return ptr; }
15
16  private:
17      T* ptr;
18  };
19
20
```

Buffer/Shader helper class

I have also created a class that can be called to initialise the buffers and shaders needed across several classes so that I would not have as much duplicate code.

Shader_Intigration.cpp

chestMovement (Global Scope)

```cpp
#include"Shader_Intigration.h"
//--------------------------------------

HRESULT BufferInitialisation(ID3D11Device* device, UINT sizeOf, ID3D11Buffer** constantBuffer, ID3D11SamplerState** sampler)
{
    HRESULT hr = S_OK;

#pragma region Constant Buffer initialisation

    D3D11_BUFFER_DESC constant_buffer_desc;

    ZeroMemory(&constant_buffer_desc, sizeof(constant_buffer_desc));
    constant_buffer_desc.Usage = D3D11_USAGE_DEFAULT;
    constant_buffer_desc.ByteWidth = sizeOf; //Number of bytes in buffer0 struct
    constant_buffer_desc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;

    hr = device->CreateBuffer(&constant_buffer_desc, NULL, constantBuffer);

    if (FAILED(hr)) return hr;

#pragma endregion

#pragma region Create Sampler State

    D3D11_SAMPLER_DESC sampler_desc;

    ZeroMemory(&sampler_desc, sizeof(sampler_desc));
    sampler_desc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
```

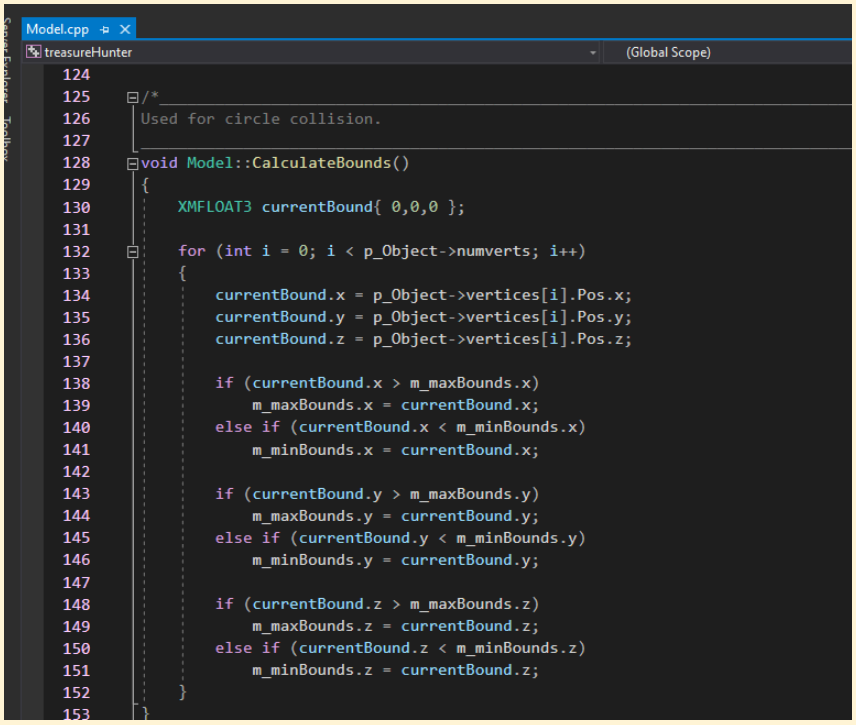
Window Resized

The window can be resized and played in different sizes.

```cpp
#pragma region Create Window

    int xPos{ 100 }, yPos{ 20 };

    hInst = hInstance;
    RECT rc = { 0, 0, 1280, 720 };
    AdjustWindowRect(&rc, WS_OVERLAPPEDWINDOW, FALSE);
    hWnd = CreateWindow(
        Name,           //Name of class the window will use
        WndName,        //Title displayed in the window's title bar
        WS_OVERLAPPEDWINDOW,
        /*CW_USEDEFAULT,
        CW_USEDEFAULT,*/
        xPos,
        yPos,
        rc.right - rc.left,
        rc.bottom - rc.top,
        NULL,
        NULL,
        hInstance,          //Handle to instance. Same value from WinMain(). Returns g_hWnd
        NULL);

    if (!hWnd) return E_FAIL;

#pragma endregion

    ShowWindow(hWnd, nCmdShow); //Takes window handle g_hWnd and displays it

    return S_OK;
}
```

Sphere Collision

I added sphere collision but did not use it, as I preferred box collision for my game.



REFLECTION

During the production of this game, the design changed quite dramatically however, the core concept stayed the same. The player is still a pirate searching for treasure (albeit with a very primitive model) and the objective is the same – dig up the treasure and push it to the ship's gangway. There were a few game mechanics that were never implemented though due to the scope being too large for my current skillset, and I had to instead replace the bottle pick-ups with a shovel pick-up, which the player requires in order to dig. With that in mind, my test plan also had to change, which meant I could only test against the mechanics that I intended to implement in the final design. I think overall, it went well though and I enjoyed learning how to use DirectX.

REFERENCES

BEEPA PTY LTD, 2022. *FRAPS* Available from: https://fraps.com/

CPLUSPLUS, 2022. - *C++ Reference* Available from: https://www.cplusplus.com/reference/chrono/

GEEKSFORGEEKS, 2021. *Smart Pointers in C++ and How to Use Them* Available from: https://www.geeksforgeeks.org/smart-pointers-cpp/

SHAARIGAN, 2022. *Limiting framerate in a game (using std::chrono)* Available from: https://www.gamedev.net/forums/topic/711573-limiting-framerate-in-a-game-using-stdchrono/5444737/