Desert Wanderer - The Game

Group members: Luke Jordan and Nick Bunge

Project Overview

Project Goals:

We wanted to create a simple platform “game” where we have sprite animation and the player is able to jump onto a platform. Each action has their own animation.

Project Description:

First we started with animation. We wanted to start with a simple idle animation to prove we could produce a sprite with a functioning animation, and then we would add more animations once we figured out how the sprite would idle. As Luke was working on that, Nick worked on setting up the platforms and building boundaries so the player could walk around the scene. Nick also set up the textures for the platforms with a single block texture that was tiled across the entire mesh. After we finished this, we moved on to changing the animation based on which way we were moving and if we were jumping or not. We then fixed a few bugs with how the movement and the animation was working and finally added background and foreground textures at different z positions to create parallax to simulate a 3d environment. Finally we added a camera that would follow the player to the end of either side of the scene.

We used vs code for our development environment, and we used the skeleton code from assignment 6, but an updated version of Sean Barrett’s stb_image.h. This includes GL and GLFW. The program can run on a linux environment: specifically the Clemson CPSC lab machines. The program works both through the FastX virtual machine or the actual lab machines.

When we first checked the physical lab machines to see how the program ran on those, we noticed that the change frames per second really messed with our program, but we then figured out that we can turn v sync on so it will always run at 60 fps. This makes the movement and the animation run at a set value all the time on any machine. Because different animations have different number of frames, and we didn’t want the jump animation playing the same as the walk animation, we had to find a way to change the number of frames in the animation, reset the animation based on the new action, and also pause the frames like when jumping. The jump has a pause in the middle before descending.

Algorithm for changing animation speed/pausing for jump

```
now_time = glfwGetTime();
time_delta = now_time - old_time;
bool down = true;
_update_fps_counter(g_window);
now_time = glfwGetTime();
time_delta = now_time - old_time;
if (!jumpTrigger && !fallFlag)
{
    frames_ps = 8.0f;
    jumpCount = 0;
    if (time_delta >= 1.0f / frames_ps)
    {
        old_time = now_time;
        time_delta = 0.0f;
        uv_x += 1.0f;
        if (uv_x >= numXInAnimation)
        {
            uv_x = 0.0f;
        }
    }
}
else
{
    frames_ps = 4.0f;
    if (time_delta >= 1.0f / frames_ps && uv_x <= 4)
    {
        old_time = now_time;
        time_delta = 0.0f;
        uv_x += 1.0f;
    }
    else if (jumpCount <= 125)
    {
        jumpCount++;
    }
    else
    {
        if (time_delta >= 1.0f / frames_ps)
        {
            old_time = now_time;
            time_delta = 0.0f;
            if (uv_x < numXInAnimation)
            {
                uv_x += 1.0f;
            }
        }
    }
}
```

Algorithm for moving left and right on the ground or in the air, but not into the side of the platform or out of bounds

```
//moves player left and right
    if (position.v[0] + spriteX > -BORDER_X && position.v[0] + spriteX < BORDER_X ) //determines if player is in border bounds
    {
        if((position.v[0] > -PLATFORM_X && position.v[0] < PLATFORM_X && position.v[1] < PLAT_TOP_BOUND && position.v[1] > PLAT_LOW_BOUND) && !fallFlag && !jumpTrigger)    //determines if player is in the platform
        {
            if(abs(position.v[0] + PLATFORM_X) <= abs(position.v[0] - PLATFORM_X))  //determines which side the player is closest to and moves them out of the platform
            {
                model_player = translate(model_player, vec3(-movementSpeed, 0.0f, 0.0f));
            } else
            {
                model_player = translate(model_player, vec3(movementSpeed, 0.0f, 0.0f));
            }
        } else
        {
                model_player = translate(model_player, vec3(spriteX, 0.0f, 0.0f));
        }
        if (position.v[0] > INITIAL_CAM_BOUNDS_X || position.v[0] < -INITIAL_CAM_BOUNDS_X)
        {
            if (isD && !isA)
                moveCameraRight(view_mat);
            else if (isA && !isD)
                moveCameraLeft(view_mat);
        }
    }
```

[Working program video]

Experiments

Sprinting

We attempted to implement a sprint function that would run faster after a set time of walking. We could not however get this to work with our camera movement we had already implemented. We decided to scrap the idea and instead moved on to parallaxing and creating a 3d environment. We first attempted a sprint that was controlled by a key press. Handling the key press events along with movement, jump, and camera movements events seemed was a problem that could be solved with more time than we were going to commit to. Another one of our attempts was to active the sprint after walking in a direction for a little while. This proved to also break the camera movement we had, and we decided to spend our time working on other features. Overall we learned that its good to have a modular code base design, so adding new features won't break as many previous features.

Camera Movement

When we were creating our movement and the environment we were in, we thought it would be a nice touch to add a camera that would move across the screen with the movement. This was no easy task, and took us a few tries to get it right. For a while we were having troubles with the camera not becoming centered again, and being offset. We found a way to fix this where we decided we were moving the player and what that looked like combined with where we were moving the camera.

Sprite Animation

Throughout the project, we changed what sprite set we were using a few times. At first we had multiple sprite sheets that would have been a pain to constantly change the texture of an object, so we decided to get a new sprite sheet, one that was all together. This one looked nicer, but there was something off. The sprites were offset because the spritesheet was not a uniform size above and under all the sprites. In the end, I had to do a lot of editing of the sprite sheet to be able to have it a uniform size and padding between all of them. We learned that the sprite sheet size matters just as much as the math you are doing to split the sheet up into individual frames.

Idling

We ran into a problem when first implementing the movement. The character would have a walking animation, but they would constantly have a frame of idling animation. This had to do with the way OpenGL handled key hold events. When a key is held it constantly produced key press and key release events that we were unaware of. Because of this our animation would continue to break. Our solution was to have a counter that started once a key release event was initiated. If the counter was able to count up to 20 cycles (in the rendering loop) without another keypress event than that player was considered idle, and the idle animation would run. This problem helped us getting a better understanding of the key action events in OpenGL.

Conclusion

In conclusion, this project was deceptively difficult. Luke had already done some animation in Unity, the game engine, so he thought that it would be pretty simple. It was more difficult to implement than expected. The lack of automatic sprite splitting and trying to set up the timing for states like the jump were the biggest hurdles when it came to the actual animation. However, the previous 6 assignments did help tremendously in speeding up the production of our platformer. We also were able to implement techniques that were not addressed in the assignments like sprites, animations, or parallax backgrounds.