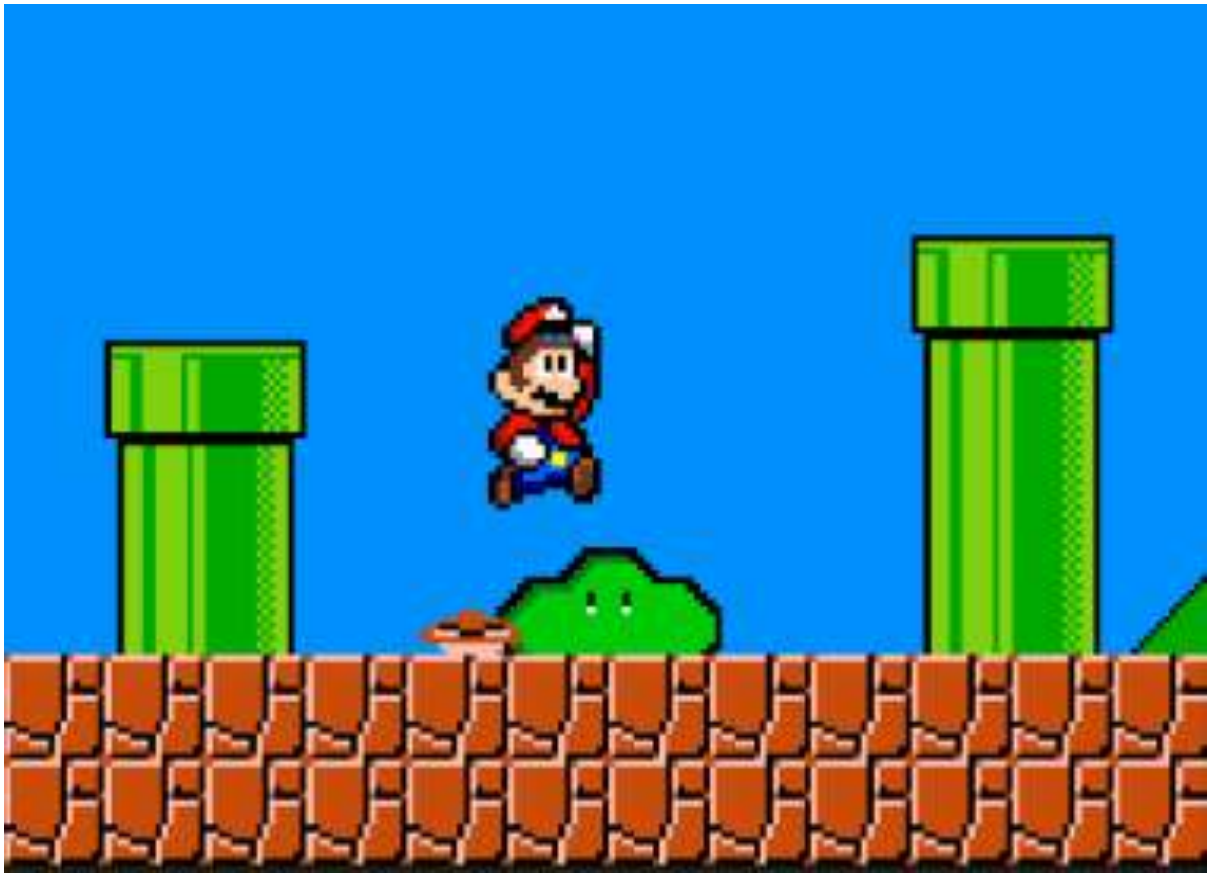
Super Mario Bros NES Unity Recreation

17th August 2025



Background:

In June of 2025 I set out to create my first game in the Unity game engine. Before this project the only game I've made was 21 Bust / Blackjack in Visual Studio. Making this game was going to challenge and teach me on how to make more complicated games and learn the basics of 2D game development with a proper game engine used by game studios. I wanted to start with something basic and simple to make for my first game so that I can learn to how to use the Unity game engine and all its features and components in order to learn to how to us it and gain knowledge on how to make future games with my new experience and learning from my mistakes in order to make a better game in the future.

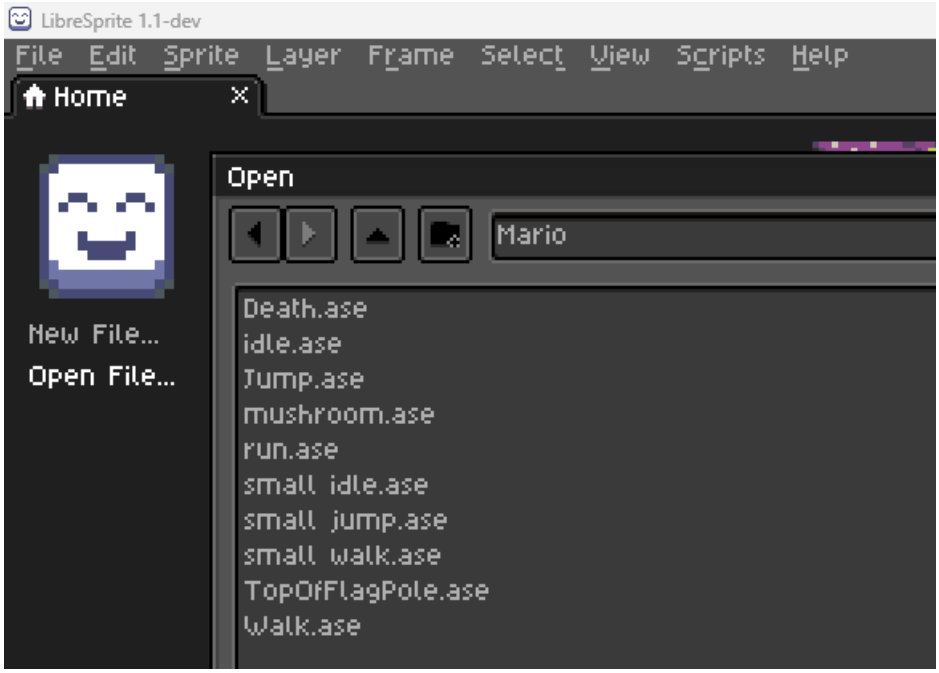
When I first tried making this game, I followed a YouTube tutorial on how to remake Super Mario Bros NES, how that was a big mistake as it led to random bugs and errors that I couldn't fix, so I decided to restart the project from scratch and used some stuff I learned from the video such as how prefabs, scripts, 2d colliders, and more works in Unity in order to make the game on my own. When developing my game, I would different videos and websites to learn different parts of Unity as this helped me to learn to use unity instead of learning how to recreate 1 game, and this gave me the skills and knowledge to learn how to make different games using the software instead of only knowing on how to recreate one game.

Planning Stage:

Before I created the game, I first set out a list of idea for what the game can have which include:

- Recreate the level 1-1 from Super Mario Bros NES
- Create my own Sprites for the game
- Custom player movement
- Super Mushroom to make the player go from small to big Mario
- Have Goombas and Koopas with their own Ais
- Warp pipes to bring you to a different area
- Flag pole to end the level

Sprites:



I used the pixel art software LibreSprite to create the sprites for my game.

When creating my Mario Sprites for my game, I tried recreating the sprites from Super Mario Bros 3 in the style of the modern-day design of Mario.

After I made a few sprites for my game, I ported all the sprites into my sprites folder in Unity in older to use them in Unity as prefabs for my game.

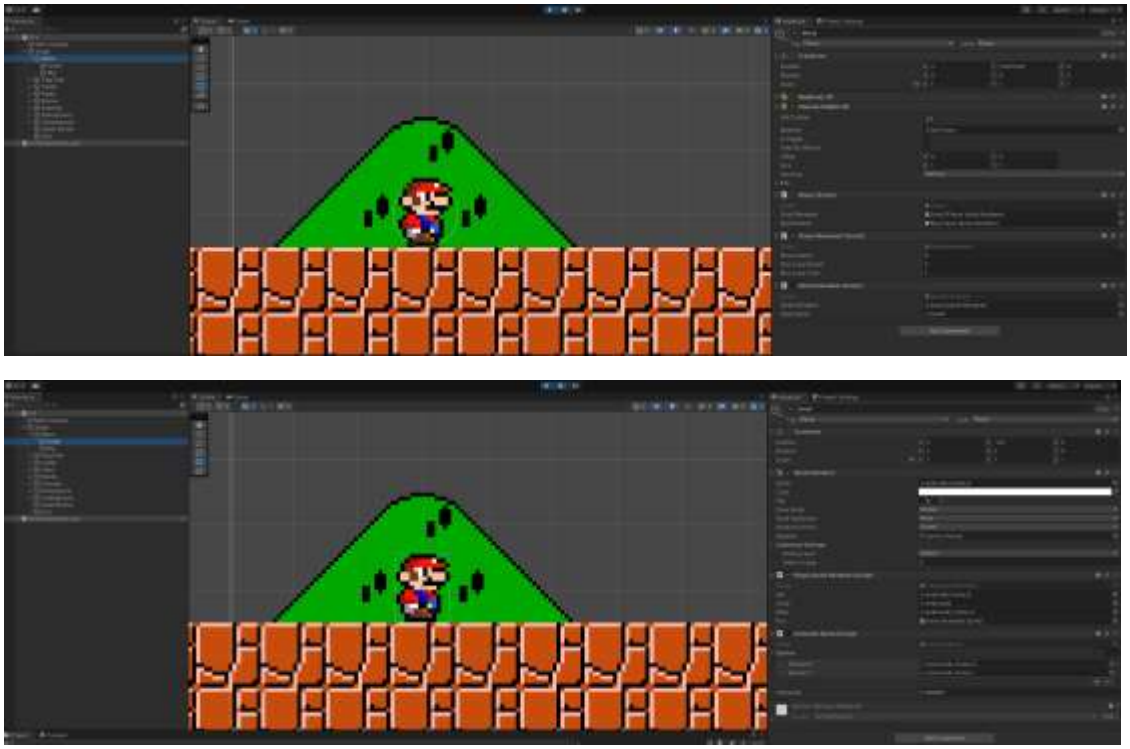
Prefabs:

After importing my sprites into Unity, I converted them into Prefabs / reusable objects I can easily place throughout my game. These include the player, terrain pieces, background elements, enemies, and more. Each Prefab has its own set of properties, for example, the player and terrain objects use 2D colliders so they can interact properly, while background elements have no colliders, allowing the player to move through them without obstruction. This setup helps keep my project organized and ensures each object behaves as intended.

Level:

When building my game's level, I created a dedicated scene to hold the entire layout. Using my terrain, blocks, pipes, and background element Prefabs, I reconstructed the layout of Level 1-1. Each Prefab had to be placed at precise coordinates so everything aligned correctly, and I also adjusted their 2D collider positions to match their visual placement, ensuring smooth movement and accurate collision for the player. After finishing the level layout, I positioned the player and camera at the starting point and placed all enemy Prefabs at their designated spawn locations.

Player:



When creating my player, I built a Player Prefab that holds all of the player's core properties and scripts, including movement, jumping, power-ups, a 2D collider, the starting position, and more. Since the player has two different forms with being small and big after collecting a power-up, I created subclasses nested within the main player class to manage these states consistently. Each state uses its own collider size that matches the corresponding sprite, ensuring that both the small and big versions of the player function correctly while maintaining accurate collision boundaries.

I created several scripts to control my player's movement, using different variables and calculations to handle acceleration, running speed, jump height, deceleration, and fall speed. These behaviours were achieved by calculating velocity, player position, gravity, and time, allowing each value to smoothly update during gameplay. By fine-tuning these calculations, I was able to make the player feel responsive while still having a sense of weight, resulting in movement and jumping that feel natural and satisfying to control.

Testing:

During the testing phase of my game, I encountered several issues and glitches that needed to be fixed. One problem involved the camera moving both right and left, even though the original game only allows rightward scrolling. I resolved this by adding code that prevents the camera from scrolling to the left.

Another issue occurred when the player jumped and hit a block from below. Instead of stopping and falling, the player would continue the upward jump motion and hover under the block until the jump arc finished. To fix this, I added logic that detects when the player's collider hits an object above them and immediately reduces the jump height, causing the player to drop correctly instead of floating.

While testing player movement, I spent time running and jumping around the level to fine-tune the character's feel. I made small adjustments to movement speed and jump height until everything felt responsive and natural. This process helped me balance the player's momentum, giving them enough weight to feel grounded without making the controls feel sluggish or floaty.