California State Polytechnic University, Pomona

Doodle Jump Recreation in C

Lucia Maturino Iniguez

Marco Joson

Ethan Nhan

Chelsea Kathleen Ocampo

Branden Romero

CS2600: Systems Programming

Professor Danica Cariaga

11 May, 2025

Introduction

Given our collective experience with programming in C as well as the time constraints given by academia, work, and personal lives, we wanted to recreate a classic game from the mobile game era in the 2010s, Doodle Jump. This game allows us to create a MVP (minimum viable product) quickly and have lots of room for expansion and extra features. Despite the simplicity of the game and code, implementing everything and working with SDL for the first time proved to be a challenge. In the end we were able to create a complete and well-functioning Doodle Jump recreation for our CS2600 final project.

Development

Starting development, we researched different methods to output a window where our doodle jump game could be tested and played. Ncurses was our first option, but we found a better option. We settled on using the Simple Directmedia Layer (SDL) library, which would be the most efficient since it can be used in Visual Studio. In order to collaborate on the project we also used Github to share our code within a repository. Within the repository, we created other branches apart from main so as to not disturb the program from the main branch until it was safe to merge branches.

Setting up SDL in Visual Studio, we mainly followed a YouTube tutorial to link it to the program (Medina). In the tutorial, the youtuber initially used an example from the SDL3 website called “clear” featuring a window that changes colors, to check if the program is working in Visual Studio (“SDL3 Examples”). After we finished setting up SDL, we felt that the “clear” code fit our game as a background, and decided to use the code as a template to start our project on.

Beginning the program, we created an SDL window and renderer using static SDL_Window *window = NULL and SDL_Renderer *renderer = NULL. Then we created a function that runs once per frame that changes color by clearing the window with SDL_RenderClear(renderer). And presenting the different color every frame with SDL_RenderPresent(renderer).

With the window set up, we began implementing the player. We used both documentation and examples from the SDL website to research ways to implement the mechanics that we wanted (“SDL3/FrontPage”). For implementing the player, we used mainly the snake game as a reference for building the structs, using AppState to store things between frames, input handling, and rendering the box for the player (“SDL3 Examples”). Using previous knowledge from game engines, we easily implemented player kinematics to the game such as applying velocity to the player’s position and applying jump force and gravity to the player’s velocity every frame.

An issue that we had is that the inputs were acting oddly — whenever you hold the right or left key, the player moves as if only a press of a button was inputted, then a second later, acts correctly. It turns out that the Key Down Event that we are using to detect key inputs does not handle holding down keys very well. By using the Key Up Event along with the current implementation, we were able to fix the issue.

After the player, we started implementing the platforms using the same methods to develop the player. Within the function that updates the player called update_Player, we included interaction with the player and platform. A “for” loop checks if the player's y velocity is greater than zero for every platform (the maximum number of platforms is ten). If it is, the player will be able to enter from underneath the platform and bounce off the top. When the player reaches the middle of the screen the platforms move down to give the illusion that the player moves up.

Platforms are randomly positioned so the player does not have one way to navigate the terrain. To figure out when the game ends and player loses, a function calculates when the player's y position is less or equal to the screen's height, which will then reset

Having the core functions of Doodle Jump, we sought out to do extra mechanics. One of these mechanics was a main menu at the start of the game and a menu every time the player dies. To implement this, a button struct was used to keep track of its x, y, size, and text that would be used for the given button. The SDL_PointInRectFloat function is used to detect whether the mouse is pressed down within either play/quit buttons. When the app is first launched, the main menu creates a backdrop with the game title, start button, and quit button. If the user presses the green button (start button/play again button), the reset_game function runs, initializing the player and platforms, and begins rendering the game. If they press the red button (quit button) the window closes using the SDL_APP_SUCCESS event. Once the game has started and the player loses, the last rendered game screen is kept and a small menu pops up with a retry and quit button. The score is displayed at the top of the small menu, and just like the main menu, a clicking on the retry menu restarts the game while the quit button closes the app.

To calculate score, the AppState stores an integer that is reset each time the game starts or restarts. The score is updated and increased each time the player moves up, all within the if statement that checks if the player is at half height the score adds ten points to the player. The score does not reset to 0 after the game ends until the reset button is chosen. This allows for the screen and menu to display the final score the player got instead of making it 0.

Lastly, we added some finishing touches to the game to make it more aesthetically pleasing and be more user-friendly to the player. After the player presses the "play" button in the start or retry menu, we implemented a pause to show a small tutorial text to show that the

movement controls of the game are through the “A” / “D” key buttons or the arrow keys which was implemented by simply introducing a has_player_started variable in the Player struct. Once the player presses any buttons, the game will start moving and the tutorial text disappears. We also added a font other than arial that fits the casual fun vibe of the game more, as well as fixed or adjusted spacing in some of the text in the game, which includes the different screens, and the buttons.

Strengths/Weaknesses

Throughout the project, we were able to communicate well in order to continue working on different sections of the code. When certain bugs arised, everyone in the team was able to exchange ideas and solve the issue at hand efficiently. It being the first time we used the SDL library, we were able to learn and apply the SDL library to our code. The process of figuring out how to use SDL ended up taking much longer than initially expected. There were numerous different variables that were required to be used in order to just make a window appear. On top of that, there was a certain order of code that needed to be done to create the desired display. Of course, this also resulted in setbacks, while we often referenced the SDL website for help or to find certain methods that we could implement in the game.

Even though SDL gave us a lot of tools for us to use to make the process of building our ideal vision easier, with the help of multiple libraries of resources, we could use, which led to some weaknesses with our project. One of them being that if one of the libraries, for example, the TTF library, wasn’t installed correctly or not correctly located on your IDE, it can lead to multiple points in which some part of the game won’t work, won’t display, or even worst, would crash on launch. So the main weakness we can glean from our work is how important the dependencies and making sure everything is located and called in the right places, and how it

requires multiple different libraries to get simpler functions to work. But by using multiple different libraries meant it was a weakness, it also became one of our strengths. Being able to use multiple libraries lets us have a lot more control over how we want everything to look and function. For example, the TTF library was a struggle to get it installed correctly, but once we did it gave us so much to work with like, being able to change the font, the size, and the color of the text that not only displayed on the game main screen but also on our buttons. So, using SDL led to us having some weaknesses, but it ended up giving us a lot more strength in our project, which made it a much better product overall.

Areas of Improvement

Code: Since this is our first time using SDL to code, we all took time to ourselves to learn individually and experiment with the code itself. However, with us having to work on it separately and the fact that SDL is low-level compared to other game engines, it sometimes leads us to not be able to follow each other's code that well without asking the person responsible some questions. Sometimes it led to setbacks, while we were researching the best way to make the code function (there were not many resources available). Consequently, we often referred to the SDL documentation and ourselves, analyzing how to overcome said obstacles during development. If we have written more comments explaining what parts of codes are doing then we could have mitigated it more than what we already have. Additionally if we had more practice with SDL it may have been easier to understand the code each of us was implementing without having to explain.

Teamwork: After figuring out which library would be best to create our project, we could have begun planning and delegating parts of the project that needs to be done. The work distribution was not super organized. All of us did a task that was not picked up or unfinished.

Work on the project was basically picking up where another left off and adding on another required part that would make the game function. We could have communicated more throughout the project, checking in on others to see if they already finished their task. Other than that, we were able to stay in contact and reach out to other teammates when we needed help or suggestions to a problem. We were mostly careful when we merged branches with main and mentioned when we would merge within the group.

Contributions

Lucia Maturino Iniguez: Score calculation, initial text display, platform resizing

Marco Joson: Updated player movement, platform creation (drawing, x-axis movement, y-axis movement, collision)

Ethan Nhan: Start menu, death menu, button functions, cleaning code, scripts

Chelsea Kathleen Ocampo: Getting SDL to work, window creation, player movement + physics (bouncing, left and right border interaction), death and respawn

Branden Romero: Adding ttf library, added text to buttons and score

Works Cited

"Creating a Game Loop with C & SDL (Tutorial)."*YouTube*, uploaded by pikuma, 22 April 2023, https://www.youtube.com/watch?v=XfZ6WrV5Z7Y&t=3906s. Accessed 28 April 2025.

Medina, Christopher. "Setting Up SDL3 w/ Windows and Visual Studio." YouTube, YouTube, 17 Dec. 2024, www.youtube.com/watch?v=Fy_4ACtmt7A.

"SDL3/Frontpage." SDL3/FrontPage - SDL Wiki, wiki.libsdl.org/SDL3/FrontPage. Accessed 10 May 2025.

"SDL3 Examples." SDL3 Examples, examples.libsdl.org/SDL3/. Accessed 10 May 2025.