17/05/2023

CI517 – Game Engine Fundamentals: Animation Playback

Module Leader: Almas Baimagambetov

Lynx Button

Student num: 21806200

Table of Contents

[Introduction 1](#_Toc135231228)

[Links 1](#_Toc135231229)

[Github Repository 1](#_Toc135231230)

[Youtube Video 1](#_Toc135231231)

[Implementation 2](#_Toc135231232)

[Animation Class 2](#_Toc135231233)

[Game Demo 8](#_Toc135231234)

[Critical Review 10](#_Toc135231235)

[Good aspects of the subsystem: 10](#_Toc135231236)

[Improvable aspects of the subsystem: 10](#_Toc135231237)

[Estimated Grade 11](#_Toc135231238)

[Conclusion 11](#_Toc135231239)

[References 11](#_Toc135231240)

[Articles 11](#_Toc135231241)

[Assets 13](#_Toc135231242)

# Introduction

Game engines are software development programs often used to assist and manage game creation. They are vital within the games industry and lay down a basic framework for game developers to expand off and create games easier than if they had begun from scratch.

XCube 2D is a simple game engine that has been developed using SDL 2.0 and C++. Within this project, the XCube engine will be extended to include an animation playback system which uses CSV files to display and play different animations on screen.

# Links

## Github Repository

<https://github.com/lynxbutton/CI517-Engine-Fundamentals>

## Youtube Video

<https://youtu.be/tPABEFc4tds>

# Implementation

## Animation Class

The animation class contains all the needed attributes for developers to access and use this animation subsystem. Each animation needed within the game will be an instance of this class, with access to all necessary functions to render smooth animations.

Each animation that is created by the user will need to have a specific CSV file which states all the necessary details of the animation in question. A CSV file is used within the subsystem to allow the user to manipulate their animations without changing the subsystem itself. It also allows for animation detail data to be easily changed when debugging an issue. The CSV file for each animation must contain the following data:

1. Animation name.
2. Frame rate.
3. Playback time.
4. Image location.
5. Number of sprites within each row & column.
6. JSON file location, if needed.
7. Sprite width & height.
8. Scale width & height.
9. Which way to flip the animation, if needed.
10. Whether the animation is to be looped.
11. All keyframes.

When choosing a file format for the animation details, the two main options which were considered was csv files and JSON files. Both file formats have distinct strengths and weaknesses:

* JSON files:
  + Typically, easier for a human to read.
  + Able to include data types.
* Csv files:
  + Consume less memory.
  + More compact.
  + More secure.

Due to csv files specific strengths for performance, it was chosen overusing a json file. However, csv files are generally much harder to read which can lead to more syntax errors within the file. To help with this a template csv file was created and placed within the csv file location for developers to access and use as reference when planning out their animations.

Text

Description automatically generatedWithin the Animation class two functions are used to read and assign values from the given CSV file. The function readAnimCSV uses the C++ library fstream to loop through all the file’s contents and return them as an array. The function will report back an error to the console if the file cannot be opened or there is more than one row within the file.

The next function, assignCSVContent, uses the content array from the last function to initialise all variables within this instance of the Animation class. The main role of this function is to check the file content for any errors which may make the animation unable to function properly. These error checks include:

Figure 1: readAnimCSV function.

* Ensuring all integers can be converted from strings and that they are bigger than 0.
  + This is completed for the frame rate, playback time, keyframes, sprite amount, sprite width and height.

Text

Description automatically generated

Figure 2: Convertion error checks for assignCSVContent.

* Ensuring locations are valid by checking if the file extension is supported.
  + This is completed for the animation sprite sheet location and when checking for a JSON file.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3: File extension error checks for assignCSVContent.

* Ensuring enough keyframes have been added to the file by comparing the amount of them to the playback time.

Text

Description automatically generated

Figure 4: Amount error checks for assignCSVContent.

If any of these errors occur, the animations ‘run’ variable is set as false and the animation cannot be used until these issues have been fixed. Each error also outputs to the console so, debugging can be efficient and easy for the developer.

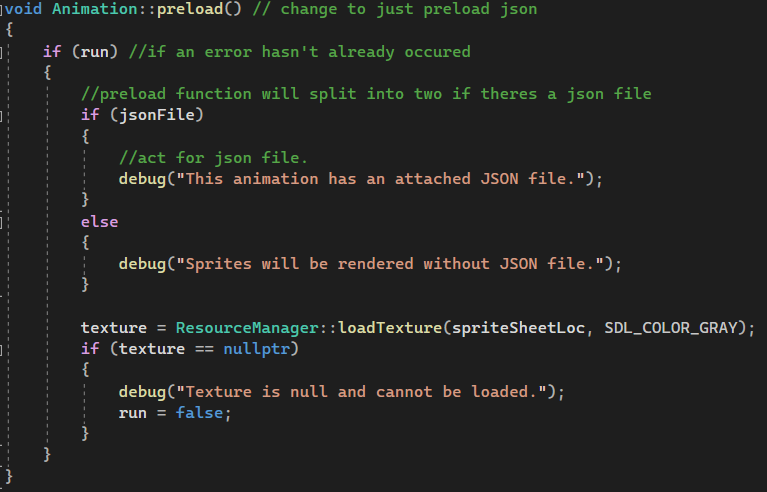


Figure 5: Preload function.

The preload function is used to prepare the needed files for rendering later in the game’s runtime. This function only needs to be ran once for each animation when the game is initialized. When preloading, the function checks for a possible JSON file needed for the animation. If the sprite sheet lacks a JSON file, then it is prepared for sprites to be split up in a grid. As there wasn’t enough time to implement JSON file support, the function declares the presence or lack of JSON file to the user and assumes the sprite sheet will still be split up in a grid-like fashion.

CubeX 2D’s resource manager is then used to load in the animation’s sprite sheet before ensuring that there have been no errors leading to a null texture.

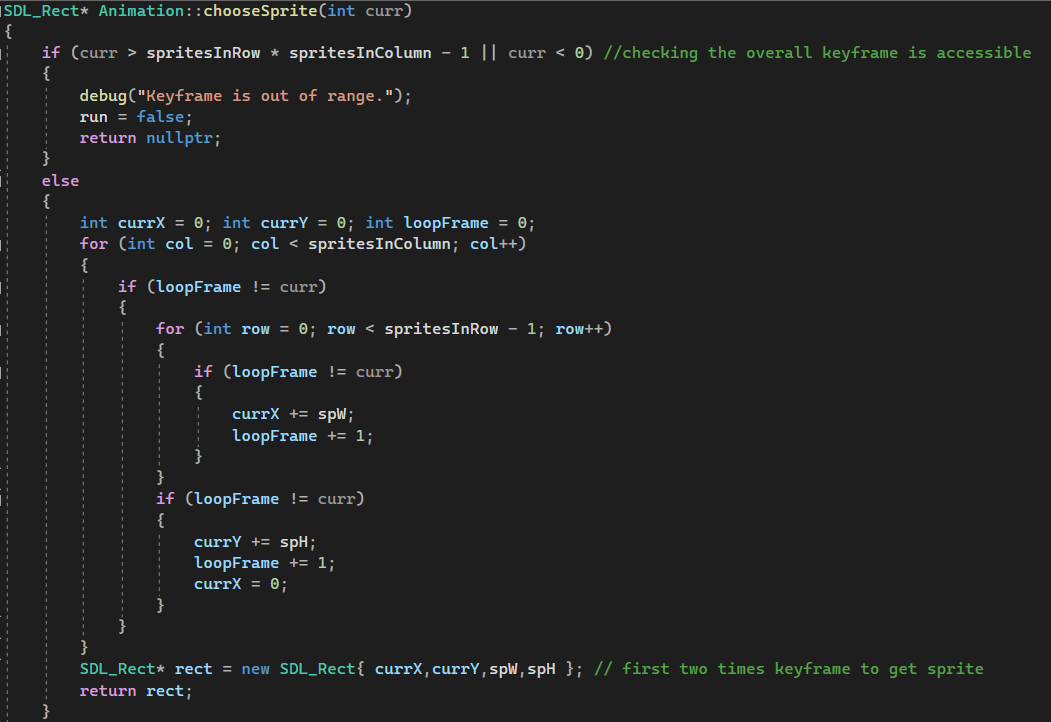


Figure 6: ChooseSprite function.

The function chooseSprite is used when the sprite sheet doesn’t have a corresponding JSON file to locate the sprites within the image. The function starts by taking the current frame and making sure that it is not outside the range of the sprite sheet. It uses the number of rows and columns known within the sprite sheet to loop through the top-left corner of each sprite till the wanted frame is reached. Once the frame is reached, it uses the coordinates of the point found and the raw sprite size to return an SDL rect which can be used to render the frame.

The function changeSprite calculates how and when certain sprites of the given animation are shown, it also controls how animations loop.

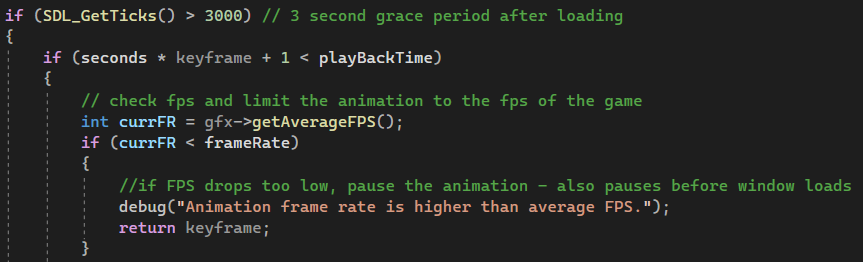


Figure 7: beginning of Change Sprite function.

To begin the function changeSprite retrieves the amount of time which SDL has been initialised and ensure it has been running for at least three seconds. This is to prevent the animation from running before the framerate has stabilised, leading to the animation being faster in the first few seconds it is rendered.

When tackling how to stabilise the overall frame rate, two main solutions were thought of:

* An initial grace period where no animation can take place.
* Locking the program at 60 frames per second.
* Locking the program at 30 frames per second.

Each solution tackles the problem in a slightly different way. Locking the game’s frame rate to a specific number of frames assists the fluidity of the frames as it evens out the peaks and valleys in the framerate, making the animation within the game keep their smooth look. However, locking the frame rate to a value higher than what a user’s machine is capable of can lead to the program being possibly unusable and unadaptive to the specifications of the machine it is being run on. As well as this, locking the frame rate to a value lower than what a user’s machine is capable of can limit animation smoothness. Locking the framerate to a specific rate is best left to user preference as it depends upon the user’s hardware as how well this will work.

Instead of this a grace period was implemented as it allowed for the frame rate to naturally stabilise before animation begins to render. Animation rendering is begun based upon the current frame rate, therefore locking the frame rate wouldn’t assist hugely when the game was starting up as the overall frame rate would still be lower. On top of this, a typical user will not notice the grace period due to other game features such as a title screen which users would reach before playing the game.

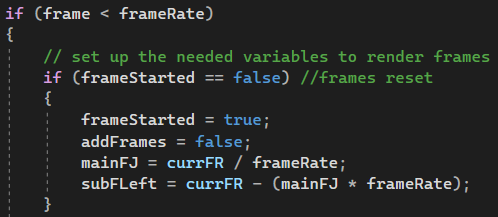
The function then ensures that the current frame rate of the screen is bigger than the frame rate of the animation. If the frame hasn’t been started the function calculates the number of frames that need to be held to achieve the animations framerate.

Figure 8: Change Sprite - Frame reset.

It also calculates if there are any remaining frames which need to be accounted for later. Using these figures, the function uses a nested if statement to switch between the base number of frames to hold a sprite and the added value. It switches between the two values until there are no more frames left within the current framerate that remain outside of the main frame hold.

Text

Description automatically generated Text

Description automatically generated

Figure 9 & 10: Change Sprite - Add Frame System.

Text

Description automatically generatedText

Description automatically generatedOnce the max framerate has been reached, the frame is restarted and another second of animation is begun. This allows for longer and bigger animations to take place within the same animation file.

Figure 11: Change Sprite - next SECOND.

One of the last things the function checks for is if the animation has been completed. If all frames of the animation have been displayed at the current rate, it then checks whether the animation should be looped. Looped animations at this stage are completely reset and non-looping animations send a message to the console to report the end of the animation. Figure 12: Change Sprite – Loop animation.

A big focus within this function was to allow the overall frame rate to be bigger than the animation frame rate. This was important as it would allow for animations with different frame rates to be used within the same space without any rendering or speed issues, allowing creative freedom. The overall frame rate could have been locked to a specific frame rate for all animations to follow, such as 24 fps, however this would decrease performance for animations which are designed to be much slower. For example, within the game demo the protagonist has a standing still animation which is a single frame – it would be a waste of processing power if this animation was laid out as a bigger and more detailed animation for the user’s computer to deduce. This also allows for animations with very large frame rates, however the game engine will throw an error if the animation frame rate is higher than the current frame rate, so it is still important for developers to keep in mind lower spec machines when animating.

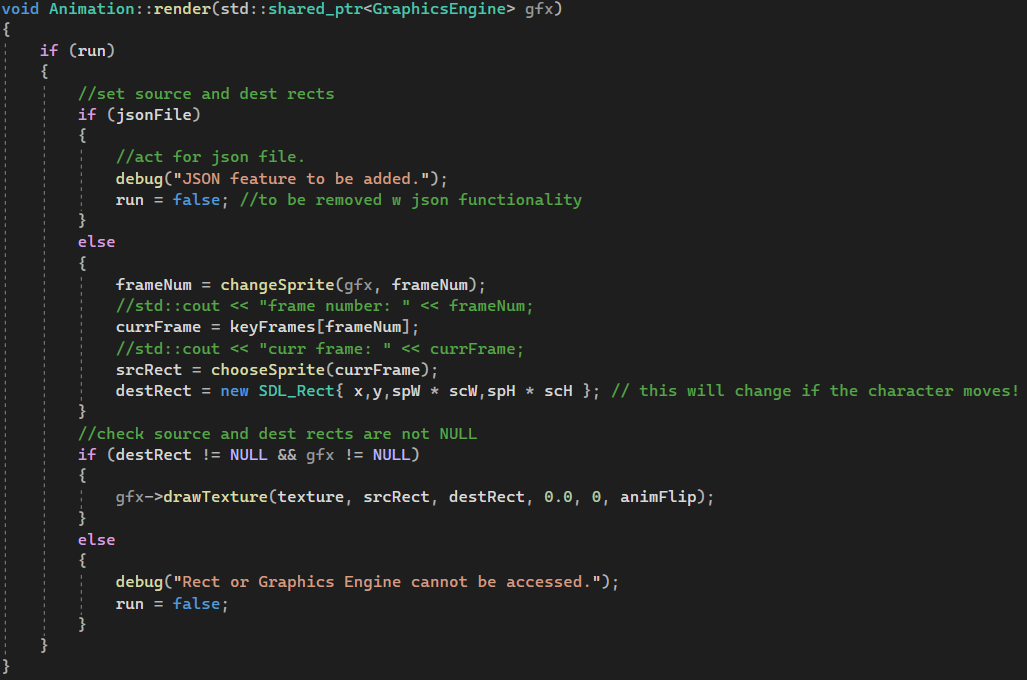


Figure 13: Render function.

The render function connects most of the animation class together by using the graphics engine to output the final animation to the screen. The function starts by checking for a given JSON file, this would be where the information within the JSON was used to select the correct keyframes from the sprite sheet. If the animation lacks a JSON file, the current frame is calculated by changeSprite and used to access the needed keyframe from the keyframe array. After this, the source rectangle for the frame is returned by chooseSprite and the destination rectangle is defined using the current x & y positions on top of the sprite and scale width and height. If none of these needed attributes return false, the graphics engine is used to render the needed sprite onto screen, else an error is returned.

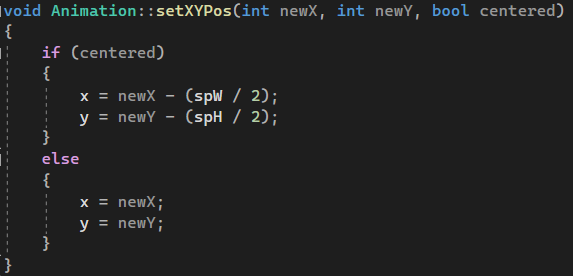


Figure 14: setXYPos function.

The function setXYPos is used within the main game file to change the location which the animation is rendered. The function gives two ways of applying the x and y positions: default and centred. Setting the function to centre divides the sprite width and height by two and takes this from the entered locations. This allows the animation to appear with the entered coordinates in the middle of the sprite. The default option within the function simply renders the animation starting from the entered location, leading the sprite to be rendered below and to the right of the location.

## Game Demo

The game demo demonstrates all the capabilities of the Animation class by having an animating protagonist which follows the user’s inputs. The walking animations of this character run at 12 frames per second for a total of 24 frames.

A screen shot of a computer screen

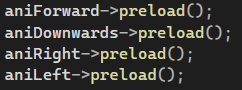
Description automatically generated with low confidenceA screen shot of a computer

Description automatically generated with low confidence

Figure 15 & 16 : Animation initialization

Within the game demo’s header file, each animation needed is initialised for use. When the SDL has been initialised, the needed information for each animation is loaded from their respective csv file by calling the constructor with the location of the needed file. Each animation is then preloaded to prepare the textures and possible JSON files for use.

With each game loop, the position of each needed animation for the protagonist is updated according to player input.

 . A picture containing text, font, screenshot

Description automatically generated

Figure 17 & 18 : Animations being preloaded and their location updated.

A screen shot of a computer program

Description automatically generated with medium confidenceWithin the game’s render function, each loop checks if the player is moving and which direction this movement is in and renders the correct animation based upon this. If the player is not moving, it uses the last direction the player moved in to render a single frame looping animation facing the given direction.

Figure 19: Animations being rendered

A screenshot of a video game

Description automatically generated with medium confidenceFigure 20: Final game demo.

# Critical Review

## Good aspects of the subsystem:

1. The system attempts to stabilise animations that are at a different frame rate to the overall game by dividing the overall frame rate by the animation frame rate. It uses this number to hold the frames for the needed time and spaces any left-over frames out to amount to the correct number of frames. This creates consistent animations which can run at the animation’s needed frame rate without risking a higher frame rate on the screen.
2. A big focus while developing this system was on tackling errors and giving detailed feedback to the developer without allowing the engine to crash. One of the ways this was done was by tackling filtering animation CSV files to attempt to ensure that only valid data could enter the system, therefore preventing possible errors.

## Improvable aspects of the subsystem:

1. The animation subsystem was designed to allow for JSON files when handling more compact sprite sheets that are not aligned within a grid. There was not enough development time to add this feature into the system however, doing so would not have been too difficult as the system is already set up for this support to be implemented.
   * To add this functionality to the system, the animation class would first need access to a library which can read JSON files, such as JsonCpp. The respective JSON file would then be parsed similarly to the animation’s respective CSV files.
   * To ensure all data is useable, digits will be ensured to be within range of the sprite sheet to attempt to prevent errors later within the process.
   * Lastly these values will be used to make sure the correct sprites are chosen within the sprite sheet with a class function which works similarly to the animation class’s chooseSprite function with use of the JSON file data.
2. While CSV files are more compact and secure this tends to matter most when a large amount of data is being used. Due to this, it may have been more accessible to developers using this subsystem if JSON files were used to provide different details for each animation. JSON files are much easier for humans to read and can hold specific data types which could’ve allowed for less complex data checking.
3. The game demo demonstrates varied animations which contain a low frame rate, such as 1, 3 & 12 however, demonstrates very little animations that use higher frame rates which developers may want to utilise. On top of this the game demo only consists of pixel art, while this is good to keep consistency within the look of the demo, it doesn’t demonstrate a wide range of how this subsystem could be used.
   * One of the ways this could be improved is by creating a more filled world with features of a pixel art RPG town. Within the world could be opportunities for more complex animations within the trees, sky, NPCs, homes, and collectables.
   * Another way is with a game demo which displays the animation system in a different light, such as a small visual novel. This could demonstrate more complex animations with different animations making up specific sprites. An example of this could be a base sprite of a human character with separate animations for their mouth, eyes, arms, etc that allow for a complex and detailed array of animations to be used.

# Estimated Grade

Total: A

Engine System: A

Game Demo: B

Report: A

# Conclusion

Overall, creating an animation subsystem for XCube 2D was a very fun learning experience which furthered my skills using SDL and C++. The system encouraged me to learn about more specific features of SDL and C++, such as the difference between source and destination rects and the best ways to use both. It also helped to solidify my understanding of references and pointers.

Throughout this project I broke down and figured out how simple animation systems work and how animations can be achieved, as well as the many different ways these problems can be tackled even when creating simple animations and features.

# References

## Articles

ClipStudioOffical (2023) Basics of the Animation Feature. Available at: <https://tips.clip-studio.com/en-us/articles/2047> (Accessed on: 26th February 2023)

Aditya Raj (2022) How to read CSV file in C++? Available at: <https://java2blog.com/read-csv-file-in-cpp/> (Accessed on: 10th March 2023)

Nayonika (2020) CSV file management using C++. Available at: <https://www.geeksforgeeks.org/csv-file-management-using-c/> (Accessed on 10th March 2023)

C Plus Plus (2022) Reference: <string>: stoi. Available at: <https://cplusplus.com/reference/string/stoi/> (Accessed on: 10th March 2023)

Mrtammy2008 (2019) How to handle stoi conversion error? Available at: <https://cplusplus.com/forum/beginner/253696/> (Accessed on: 11th March 2023)

A M (2019) Exception handling with stoi function. Available at: <https://stackoverflow.com/questions/59457656/exception-handling-with-stoi-function> (Accessed on: 11th March 2023)

Kindsonthegenius (2020) C++ String Mnaipulation. Available at: <https://www.kindsonthegenius.com/cplusplus/c-string-manipulation/#t4> (Accessed on: 13th March 2023)

Samual Sam (2020) C++ Program to Find Quotient and Remainder. Available at: <https://www.tutorialspoint.com/cplusplus-program-to-find-quotient-and-remainder> (Accessed on: 13th March 2023)

W3 Schools (2023) C++ Switch. Available at: <https://www.w3schools.com/cpp/cpp_switch.asp> (Accessed on: 13th March 2023)

Chrysanthus Forcha (2022) How do you append to a Vector in C++? Available at: <https://linuxhint.com/append-vector-cpp/> (Accessed on: 13th March 2023)

C Plus Plus (2022) Std::vector::size. Available at: <https://cplusplus.com/reference/vector/vector/size/> (Accessed on: 13th March 2023)

Perforce (2023) Game Engine Overview. Available at: <https://www.perforce.com/resources/vcs/game-engine-overview> (Accessed on: 15th March 2023)

W3Schools (2023) C++ Constructors. Available at: <https://www.w3schools.com/cpp/cpp_constructors.asp> (Accessed 16th March 2023)

Eugen Constantin Dinca (2010) C++ and SDL: How does SDL\_Rect work exactly? Available at: <https://stackoverflow.com/questions/3929920/c-and-sdl-how-does-sdl-rect-work-exactly> (Accessed on: 27th March 2023)

Game Dev Geek (2023) Moving Sprites with SDL. Available at: <http://gamedevgeek.com/tutorials/moving-sprites-with-sdl/> (Accessed on:27th March 2023)

Game Dev Geek (2023) Animating Sprites with SDL. Available at: <http://gamedevgeek.com/tutorials/animating-sprites-with-sdl/> (Accessed on:27th March 2023)

Lazy Foo’ Productions (2019) Loading PNGs with SDL\_image. Available at: <https://lazyfoo.net/tutorials/SDL/06_extension_libraries_and_loading_other_image_formats/index2.php> (Accessed on: 27th March 2023)

Gigi Labs (2015) Loading Images In SDL2 with SDL\_Image. Available at: <https://gigi.nullneuron.net/gigilabs/loading-images-in-sdl2-with-sdl_image/> (Accessed on: 27th March 2023)

The Numb (2017) Lesson 05 – Textures. Available at: <https://thenumb.at/cpp-course/sdl2/05/05.html> (Accessed on: 27th March 2023)

Tbsnsa (2021) How do the source and destination rectangles work in the SDL function SDL\_RenderCopy. Available at: <https://stackoverflow.com/questions/51908174/how-do-the-source-and-destination-rectangles-work-in-the-sdl-function-sdl-render> (Accessed on: 27th March 2023)

SDL Wiki (2023) SDL\_RenderCopyEx. Available at: <https://wiki.libsdl.org/SDL2/SDL_RenderCopyEx> (Accessed on: 27th March 2023)

SSDL Wiki (2023) SDL\_CreateTextureFromSurface. Available at: <https://wiki.libsdl.org/SDL2/SDL_CreateTextureFromSurface> (Accessed on: 27th March 2023)

Johannes Schaub (2022) How to convert a std::string to const char\* or char\*. Available at: <https://stackoverflow.com/questions/347949/how-to-convert-a-stdstring-to-const-char-or-char> (Accessed on: 27th March 2023)

Priya Pedamkar (2018) JSON vs CSV. Available at: <https://www.educba.com/json-vs-csv/> (Accessed on:12th May 2023)

Susanne Morris (2023) Data Storage: JSON vs. CSV. Available at: <https://coresignal.com/blog/json-vs-csv/> (Accessed on: 12th May 2023)

Adware (2022) Difference Between JSON and CSV. Available at: [https://www.geeksforgeeks.org/difference-between-json-and-csv/#](https://www.geeksforgeeks.org/difference-between-json-and-csv/) (Accessed on: 12th May 2023)

Ahmed Hashesh (2022) A guide to JSON using C++. Available at: <https://medium.com/ml2b/a-guide-to-json-using-c-a48039124f3a> (Accessed on: 12th May 2023)

John\_Cillis (2004) Locking Frame Rates - What does it mean? Available at: <https://www.avsim.com/forums/topic/35116-locking-frame-rates-what-does-it-mean/> (Accessed on: 14th May 2023)

## Assets

|  |  |  |
| --- | --- | --- |
| Asset Used: | Link to Asset: | Used for: |
|  | <https://opengameart.org/content/base-character-spritesheet-16x16> | Protagonist animation sprites. |
|  | <https://opengameart.org/content/grass-texture-pack-grass-texture-1jpg> | Edit for the background image. |
|  | <https://opengameart.org/content/fruit-salad-space-catch-reborn> | Collectable fruit sprites. |
|  | <https://opengameart.org/content/fruit-salad-space-catch-reborn> | Collectable fruit sprites. |
|  | <https://opengameart.org/content/fruit-salad-space-catch-reborn> | Collectable fruit sprites. |
|  | <https://opengameart.org/content/fruit-salad-space-catch-reborn> | Collectable fruit sprites. |

|  |  |  |
| --- | --- | --- |
| Splat.wav | <https://freesound.org/people/Natty23/sounds/349202/> | Splat SFX |
| bgDrums.mp3 | https://freesound.org/people/theoctopus559/sounds/624363/ | Background music |