Pac-Atac

Michael Fletcher '22, Hollis Ma '22, Daniel Wey '22, Jerry Zhu '22

Abstract

Pac-Atac is an exciting first person shooter game inspired by nostalgic Pac-Man aesthetics and first person shooter classics. Players play as Pac-Man and attempt to survive for as many waves as possible by shooting cherries and other assorted fruits to defeat the unrelenting ghosts. Developed using the Three.js Javascript library, Pac-Atac reconciles dynamic game-design and visually-exciting graphics concepts to promise an action-packed 2.5D experience in the spirit of Pac-Man.

Introduction

Goal

Our project goal was to leverage computer graphics concepts to develop an aesthetically-pleasing and dynamic game inspired by the original Pac-Man concepts. We hope Pac-Man enthusiasts and thrill-seeking gamers everywhere are pleased.

Previous Work

One past project succeeds in carrying over all elements of the 2D gameplay and instead modeling it in 3D. The player has complete information about the map similar to the original pac-man game. The rules of the game are the same as the original Pac-Man game. https://www.masswerk.at/JavaPac/JS-PacMan3D.html

Another Pac-Man related project models the game in 3D from the first person camera where the user cannot see the whole map. Additionally, the user cannot see the locations of the ghosts and where Pac-Man has traveled in the maze—limiting the gameplay experience. The aesthetics of the game are also a bit rudimentary. The rules of the game are the same as the original Pac-Man game.
https://www.playclassicgames.net/pacman-3d

Approach

We attempted to create a first person perspective shooter game inspired by Pac-Man aesthetics. The objective of the game is to survive the numerous waves of enemies (ghosts) by collecting fruit weapons and powerups on the scene to defeat the ghosts. The ideas of survival and pickups were inspired by gaming classics, such as Boxheads and CoD zombies. The game sacrifices the perfect knowledge of the original Pac-Man in place of more dynamic interaction with the ghosts. We leveraged the JavaScript library Three.js to design, develop, and render our game concept.

Our approach in creating clean, efficient code involved utilizing the render loop and modular programming, and maintaining the structure of our codebase by

refactoring and style checks. To create a real-time action game, we made use of the onAnimationFrameHandler method in JavaScript to let us simulate a smooth game. We also used classes to properly define the interactions between the objects.

To maintain our code, we used Prettier and ESLint, refactoring when necessary to enforce clarity. We initially coded everything in app.js but quickly found a need to better organize the code. Our final folder breakdown is as follows:

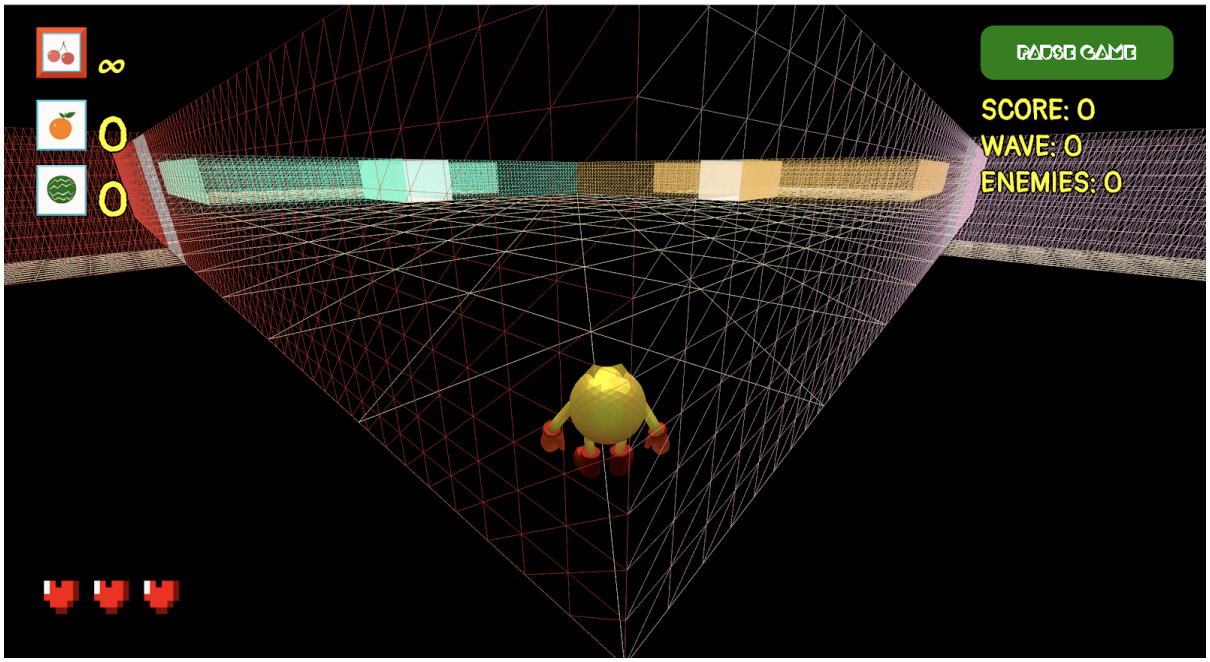
- App.js: where all the action happens
- Audio: for the music and sound effects.
- Components: for the React components used in the HUDs and menus.
- Global: global variables and constants.
- Handlers: used in the render loop to interact with the scene and objects.
- Images: background and HUD images.
- Models: meshes for the characters and pickups.
- Objects: classes that are used in our game

Our structure was successful in organizing our code in a clear manner that allows us to find files easily. This modular approach is very commonly used in programming, and it is especially powerful in games which tend to have moving characters that interact with other objects. Allowing each object to be a class makes the logic and interactions between objects more organized and easier to maintain. The render loop is also a vital part of any game, as it allows the game to frequently update and make the game smoother to play.

Methodology

Scene Layout

The scene consists of a main room with four rooms connected by hallways. Implementing hallways and rooms made the map feel more interactive and dynamic. The size of the rooms was kept relatively small to force the player to interact closely with the ghosts when dodging and defeating them.

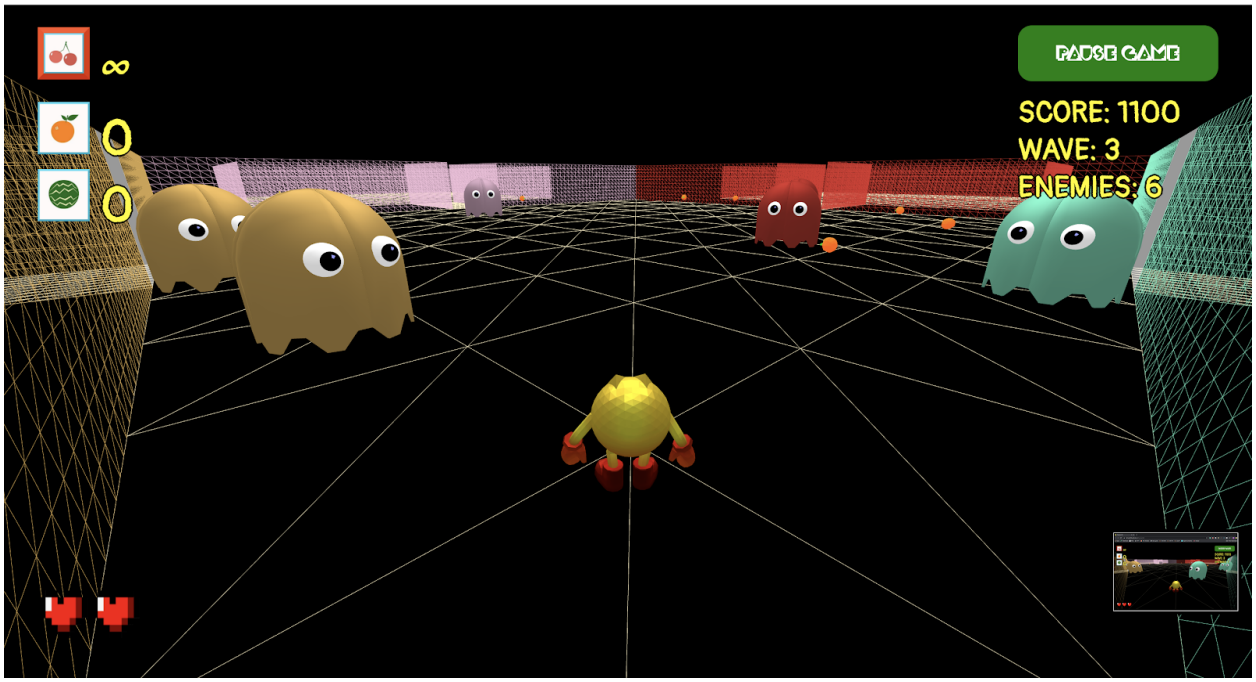


Scene Design

The scene was designed with a minimalist view in mind. The four quadrants of the map are colored in the same colors as the 4 Pac-Man ghosts. The original plan involved shaded-in walls that would be transparent. However, we faced challenges with the camera creating unintended interactions where “transparent” walls would block the rest of the scene behind it. Consequently we rolled with a see-through scene designed with mesh textures, enabling the player to see into different rooms. Designed in this way, with neon colors and a black background, the aesthetics pop and offer a minimalist and clean experience.

Ghost Mesh/Spawning

The ghost mesh was modified in blender to include only the necessary parts from its original artist. The ghosts are spawned randomly in the 5 different rooms and take on a different color depending on its location. This aesthetic choice is in line with the color of the rooms and offers an aesthetically pleasing experience.



Ghost Boxes and Superghost (Challenges)

There were two possible implementations to create a box to house ghost bosses. The first was to create a transparent box, and the second was to create a box which exhibited more glass-like behavior. This would have been done by adding cameras to the scene and setting the envmap field of the box to the output of the cameras. However, as a prototype, we implemented it with simple transparent boxes, and handled all the collisions between Pac-Man/projectiles and these boxes. Unresolvable merge conflicts forced us to scrap the code. Ultimately, the box would have been created using the second technique to provide a realistic feel to the game than simple transparent boxes, but neither are shown in the final product.

We began to implement a Superghost class, which would serve as the “bosses” to be housed in the boxes, whose defeat would signal the end of the round. We envisioned having 4 different bosses, where each team member would cook up a special set of properties for each boss. Notable ideas included bosses that spawned other ghosts nearby, bosses that faded in and out, and bosses with faster/larger properties. If implemented, these bosses could have enhanced the storyline and made for more engaging gameplay.

Scene Boundaries (pac-man & projectiles)

We employed the simple boundary checks used in Assignment 5. Whenever the Pac-Man model or projectiles would encounter a boundary, it would respectively be reset within the bounds or disappear. Each room was rectangular and thus had bounding coordinates which were used in calculations. It was surprisingly difficult to enforce boundaries in a way that felt natural to the player, especially where related to the hallways.

Round-by-round map changes

The boundaries of the map change as the player survives more waves. This was inspired by expanding map features in games like CoD Zombies. This feature makes the map more interactive for the player as they maneuver the ghosts on the map.

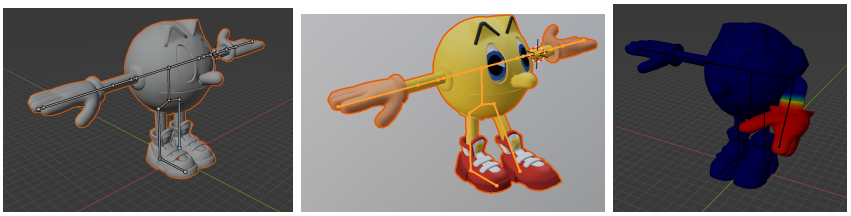
Ghost Pathing AI

A ghost's pathing behavior depends on its location with respect to the player. If a ghost is in the same room as the player, it will path in the player's direction. Otherwise, the ghost paths between pre-set "zones" in each room until it arrives in the same room as the player. This approach is simple, computationally efficient, and can manage a large number of ghosts at once.

We considered employing *Dijkstra's* shortest path algorithm but it wasn't necessary that ghosts in different rooms take the shortest path to the player. The run-time of the algorithm also seemed too costly. We also considered employing spatial hashing to avoid collisions among ghosts. However, we struggled to determine where to place the ghosts when they hashed to the same location.

Pac-Man

For deciding on the Pac-Man mesh and characteristics, there were two possible implementations. The first was to go with a basic mesh, simple in design and with a low number of vertices and faces. This mesh is in the final product that you see. The other option was to use a higher quality mesh, and, as discussed in the character animation lecture, add a skeleton in Blender which allows movement control of the mesh. We spent many hours in Blender adding a skeleton and making changes to make the visual appearance of Pac-Man's movements more lifelike (one challenge was pinching of the mesh when moving, which we attempted to handle through weight painting and what is called a corrective smooth modifier, which seeks to maintain volume through mesh transformations). However, it was decided that the mesh was still too low-quality to be animated well, and we wanted to spend more time on gameplay, so we decided to proceed with the former option. Images of our work in Blender are shown below (skeleton, movement of skeleton, weight painting):



Waves

The game spawns all enemies in waves of increasing difficulty and speed. The wave handler monitors the global property of the number of active enemies. As Pac-Man eliminates the ghosts or takes damage, this property decreases. When Pac-Man still has

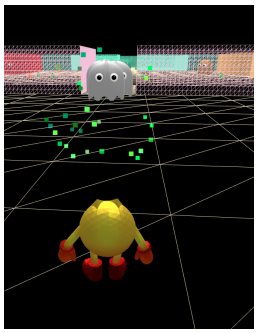
health left and there is another wave available, a countdown starts to initiate the next wave, allowing the player to obtain some breathing room. Finally, the properties of the next wave (enemy difficulty, speed, spawn rates, unlocked rooms, etc.) are loaded and the enemies spawn again.

Otherwise, upon victory or death, the game is no longer rendered, and the game state is paused. A UI shows, allowing the player the option to play again, which would free and reinitialize the game resources and reset the game state (which includes data for handling rounds). We also decided to give the player the option of choosing between having an endless game or not, allowing the player more freedom in how they want to play the game.

Shooting

All projectiles are a property of the Pac-Man object; this architecture decision would make it easier to hypothetically allow two or more players to manage their respective projectiles, despite our current version of the game using only a single character. Projectiles are created with a fruit type as a parameter, which would load the respective mesh and apply the correct properties to the fruit (projectile fired sound, travel velocity, etc.). As the fruit collides with a wall (position is out of bounds) or enemy (position is inside enemy radius), the projectile is deleted from the scene and appropriate damage is applied.

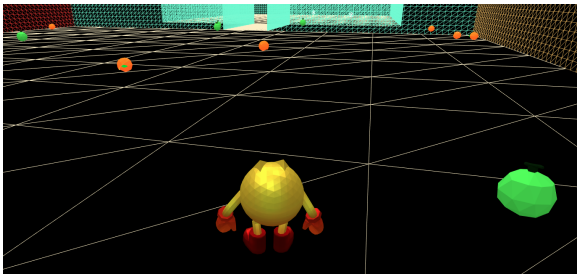
Additionally, on collision a particle explosion is generated. With the explosion color and radius determined by the type of fruit, the particles of each explosion start in the same position and travel outwards to a random point around the collision center. The size of each particle decreases as it travels outwards, giving a more realistic effect.

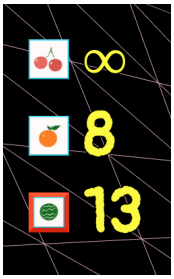


Ammunition

We chose to include an ammo feature to make the game more difficult and increase the ways to play the game. We implemented infinite cherries while having a cap for the other two fruits to balance the difficulty of the game. Infinite cherries encourage the player to play more risky and explore the outer rooms of the game, but also allows the player to spam the cherries and overcome the first couple of waves easily. Capping the ammo for the other fruits encourages the player to use their ammo wisely and to not hoard them. To implement the ammo functionality, we attached an ammo object to the

Pac-Man class and changed the object accordingly whenever Pac-Man fires or picks up ammo. We found meshes of fruits and spawned them randomly across the central room for Pac-Man to pick up.

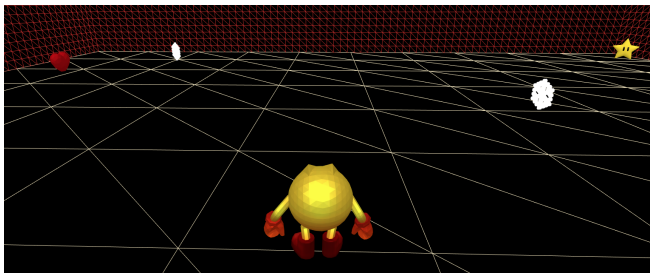


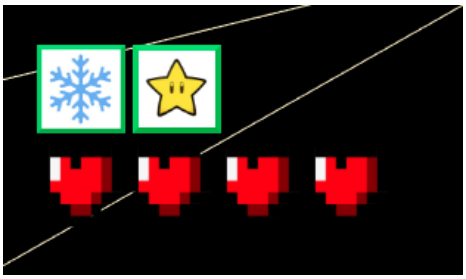


Powerups

We chose to include power ups to increase the ways to play the game. For the final waves, it becomes difficult to deal with the numerous ghosts with just the fruits alone, so power ups also allow us to take the game further than we otherwise could have. We considered adding an upgrade system, but that would require a credit system while power ups are self-contained, easy to implement, and intuitive. To implement the power ups, we found meshes for each of the power ups and spawned them in random areas in the outer rooms to encourage the player to explore instead of just staying in the central room where it's easier to kill the ghosts.

Additionally, we considered adding "bad powerups" to the scene. These powerups would result in a blur/dithering effect on the player's screen. We believe this would have been easy to implement and would represent another threat the user must navigate. However, we felt that it did not fit into our current vision of the game (these types of pickups would increase the difficulty without necessarily elevating the thrill elements we were aiming to capture).





UI/React.js

We chose to have a HUD in the game that allows the player to see their ammo count, score, wave, and the number of enemies left in the wave. Since our team members are familiar with React, we used React to create the start, pause, victory, and defeat screens. We chose to use React states instead of global variables for the game state because using state allows the React components to update.

Movement/Controls

Initially the game intended for usage of the mouse to pan the camera around. However due to complications with mouse focus on the rendered canvas and conflicts with the React UI, we found it best to use the keyboard entirely, especially since we did not want any other functionality other than camera movement to be attached to the mouse. This would also allow access to more keys during gameplay.

We originally used the keys '123' for switching between the three different fruits. After realizing that the player's right hand is most likely not pressing any keys, we moved the key-bindings to the keys 'jkl' to optimize for the player's comfort and ease while playing.

Results

How did we measure success?

Our team measured success by performing unit testing on each of the core features of the project. This increased the development efficiency and allowed for bug-resistant integration of various game functionalities.

What experiments did we execute?

The first task was to ensure that a functional 2.5-D camera was implemented with proper bounds checking of the game map. This was tested through purposely colliding the Pac-Man object against the walls of the map and ensuring that at no point the character moves out-of-bounds. Furthermore, the camera must be equidistant at a fixed angle from the flat surface of the map.

The next task was to check that the AI handlers were properly implemented after creating the Ghost enemy object. Enemies were spawned in various locations across the map and testing was conducted to check that the enemies traversed properly to Pac-Man, without entering out-of-bounds areas on the map. The next phase of the AI unit test included moving Pac-Man across different rooms while the AI was active, to ensure correct tracking of Pac-Man's location and properly executing the dynamic path

towards the position of the character. Then, damage tests were conducted to make sure that Pac-Man loses health upon collision with an enemy.

Next, projectile collisions were implemented with the aforementioned out-of-bounds and collision checking, correctly deleting the object upon collision and dealing damage if the collision object was an enemy. Powerups received similar collision testing with Pac-Man, and their respective effects on the game were tested and observed; enemies were expected to pause handling movements upon freeze while health and damage properties were tested by purposely allowing Pac-Man to collide with enemies.

UI was implemented with React, with buttons and text in each component being modified by or modifying component and game states. The React component states were often used in conjunction with the global game states, so testing involved triggering UI and in game events to see if the expected outcomes occurred. This consisted mostly of pausing and resuming the game at various points in time and determining that all game states were adjusted accordingly (Game timers move forward by the pause duration amount, game handlers were skipped and no renders were initiated, etc.) as well as using the "Restart Game" buttons to test that global game resources were cleaned up and reinitialized properly while global game states were reset properly for the next iteration.

The absence of visual bugs and error messages relating to using "undefined" and "null" variables in a non-trivial setting indicated successful game state handling.

Discussion

Overall, is the approach we took promising?

Overall, our team is satisfied with the approach we took while implementing this project. The delegation of tasks and usage of Git allowed for efficient, bug-resistant development. Abstraction was also a large component of the project after the initial commits to the project, each member was able to specialize in specific functionality without having to worry too much about the responsibilities of other team members.

What did we learn by doing this project?

We learned about game design and graphics concepts, implementing techniques and skills learned in class. By collaborating as a team, we learned that setting goals and organizing boundaries help in making a polished game and limiting misunderstandings in the workflow.

We also gained practical knowledge by utilizing Three.js in rendering our scene and making the interactions between our objects. We learned how to create scenes and implement efficient path-finding algorithms and object collisions.

In class, we spent lots of time working with meshes and 3D modeling, and this project was a valuable opportunity in practicing how to find and incorporate other people's meshes in an application. We altered many properties of the mesh including scale, color, transparency, and opacity, occasionally leveraging Blender to change properties and animate the Pac-Man model. In making the user controls, we learned about event handlers and how to create basic character movement. We increased our familiarity with the render loop, where we developed the handlers for our various components.

We also learned about web development and how to use React to overlay elements in a Three.js application, about webpack, npm dependencies, how to connect classes and objects in JavaScript, and how to incorporate sound into an application. We experimented with code stylers such as Prettier and ESLint, and had a few too many learning experiences with Git and GitHub.

Conclusion

What follow-up work should be done next?

The primary follow-up task is curating the storyline. We designed the game with a storyline in mind but lacked sufficient time to write it out. The basis of the storyline revolves around engaging with Pac-Man's past and his relationships with the ghosts he has to overcome.

In addition to a storyline, other features of future focus include implementing a leaderboard, ghosts with different properties, possible bosses, different game modes (story/survival), optimizations such as spatial hashing, additional pick-ups, a credit system, upgrades to Pac-Man (animation via adding a skeleton), and randomly generated maps.

Further work can be done outside the technical details of the project, such as marketing the game to increase its popularity, creating spin off games, soliciting user feedback, conducting AB testing to identify enjoyable/boring aspects of the game, contact psychology researchers to let them use our game as a means of testing attention or other things, etc.

What are issues we need to revisit?

At its current implementation, there do not seem to be any large issues that we have come across. On some computers, the game may sometimes be laggy (the large quantity of meshes in the scene may slow it down), which we resolve by capping the number of pickups in the scene. Despite this fix, there are some cases where lag can cause the game to behave in strange ways - such as ability to go through restricted walls.

How effectively did we attain our goal?

We're very pleased with the outcome of Pac-Atac. The project represents the culmination of many development hours, feature design discussions, and a semester of

computer graphics study. Channeling the nostalgia and fun of Pac-Man, Pac-Atac has smooth gameplay, appealing visuals, and exciting features.

Contributions

- **Jerry Zhu**
  - Initial working arena, sounds, music, projectiles, waves, shooting, projectiles, projectile explosions, HUD, menus, movement, controls, refactoring, styling
- **Hollis Ma**
  - Ammo and power ups (implementation, ideation, spawn, meshes), shooting, projectiles, HUD, menus, React, refactoring
- **Daniel Wey**
  - Scene layout, scene design, ghost AI, ghost mesh, scene boundaries for Pac-Man and projectiles, round-by-round map changes
- **Michael Fletcher**
  - Scene layout, ghost mesh and spawning, scene boundaries for Pac-Man and projectiles, (following not shown in final product) animating Pac-Man in Blender, ghost boxes, experiments with post-processing

Works Cited

Models/meshes

- Pac-Man ghost: https://sketchfab.com/3d-models/pacman-ghost-inky-bf94cb68178349f589fdb47b6c2c8b73
- Cherry: https://www.turbosquid.com/3d-models/cherry-3ds-free/760571
- Orange: https://www.turbosquid.com/3d-models/cartoon-orange-3d-model-1399269
- Melon: https://sketchfab.com/3d-models/melon-5fb92707165c4214a3127d67557fc24c
- Freeze: https://sketchfab.com/3d-models/snowflake-5cb68fa2bd1a43eca4f0fc7f5c676a8d
- Star: https://sketchfab.com/3d-models/super-star-super-mario-3d-world-c81d2a0c9d4e437c82386dbff7f6fee0
- Heart: https://sketchfab.com/3d-models/love-heart-1985fd8a99554590b728ff8b1dcfc33c

Images

- Fruit icons:

https://i.pinimg.com/originals/7b/9a/52/7b9a52fc4cf5fa4b8c73efdbdf2c2834.jpg
- Heart icon: https://art.pixilart.com/699fb46a495e3d1.png
- Star icon: https://www.pngfind.com/pngs/m/2-20080_28-collection-of-mario-star-clipart-super-mario.png

Audio

- Global music: https://www.youtube.com/watch?v=MEtdyn2QEzA
- Star music: https://www.youtube.com/watch?v=TLb33K8UO30
- Victory music: https://www.youtube.com/watch?v=owvJUr3LvB8
- Defeat music: https://www.youtube.com/watch?v=votpmwC25Ek
- Health noise: https://www.youtube.com/watch?v=6G-k4zxou7Y
- Hitmarker noise: https://www.youtube.com/watch?v=3fVfRPADqtE
- Ghost noises: https://www.youtube.com/watch?v=rg0iWdUvHDA
- Damage noise: https://www.youtube.com/watch?v=0T_NR2KY8uI
- Wave music: https://www.youtube.com/watch?v=IHb4BPCYYuM

Documentation & examples

- https://docs.blender.org/manual/en/latest/
- https://jsfiddle.net/1cv2p5aq/
- https://threejs.org/docs/#manual/en/introduction/How-to-use-post-processing
- https://threejs.org/docs/#api/en/cameras/CubeCamera
- Much more from Three.js documentation