K.R.A.B.S.

Keyboard Rhythms and Beats Synthesizer

Valerio Di Stefano (1898728)

Introduction

K.R.A.B.S. is an online music synthesizer that allows you to create beats and melodies using your keyboard.

Project's source code is available in the GitHub page at the link:
https://github.com/SapienzaInteractiveGraphicsCourse/final-project-krabs

Project is available online at this link:
https://sapienzainteractivegraphicscourse.github.io/final-project-krabs

From the website's main menu, select one of 3 different keyboard and scene styles and start using the synthesizer.

Music controls:

- Use your computer's keyboard letter keys (from A to Z) to play beat sounds
- Hold down your computer's keyboard letter keys to loop the beat sounds
- Use the "," button (corresponding to the musical note button in the 3D scene keyboard) to generate a random beats combination
- Hold down the "Alt" button (corresponding to the crab button in the 3D scene keyboard) to play the crab melody
- Click on one of the 5 colored squares to make the crab move there (and change the crab melody)
- Use the space key to stop all sounds

Camera controls:

- Use mouse left click to rotate the camera
- Use mouse wheel to zoom in/out
- Use mouse right click to move (pan) the camera
- Use Tab to toggle the camera view between the crab and the keyboard

Other controls

- Use Enter to hide the overlay controls info
- ESC to get back to the main menu

Technologies

The project was created using plain HTML, CSS and JavaScript.

Two external libraries were used:

- THREE.js (as a web 3D environment)
- Tween.js (for animation)

Other than the external resources used (imported 3D models, images for textures and icons, the font OTF and typeface JSON files, the sounds and the 2 external libraries), the entire application only consists of 3 files (stored in the root folder of the project):

1) An “index.html” file containing the main HTML document from which the web page is generated (including the canvas in which the 3D scene is drawn and the UI elements shown in the page);
2) A “style.css” file containing the style of the basic HTML elements as well as the UI elements of the application;
3) A “main.js” file containing ALL the JavaScript code written by me to display the 3D scene and assign functions to the various elements.

The “main.js” script also imports other THREE.js library modules other than the main three module, which are:

- Stats from './three/stats.module.js' (for debug purposes, used to show the FPS of the scene and other overly debug information);
- TextGeometry from './three/TextGeometry.js' (to generate the letters and symbol geometries for the keyboard buttons);
- FontLoader from './three/FontLoader.js' (to load OTF and typeface JSON files in order to be displayed as geometries using the TextGeometry module mentioned above);
- GLTFLoader from './three/GLTFLoader.js' (to load/import external GLTF 3D models);
- TransformControls from './three/TransformControls.js' (for debug purposes, to move and rotate objects in the 3D scene);
- CCDIKSolver from './three/CCDIKSolver.js' (for inverse kinematics, see later);
- OrbitControls from './three/OrbitControls.js' (for orbit controls).

In the following, I will refer to the 3 scene accessible through the 3 buttons of the main menu as:

1) “Laptop version” or “laptop keyboard” (for the scene accessible from the first button of the main menu, the orange button);
2) “PC version” or “PC keyboard” (for the scene accessible from the second button of the main menu, the sky-blue button);
3) “Typewriter version” or “typewriter keyboard” (for the scene accessible from the third button of the main menu, the pink button).

I will also refer to the 5 clickable squares in which the main crab can move (and which defines which instrument the crab will play, and thus which melody will be played) as “crab bases”.

External Tools

External tools used (to create and edit various resources used in the website):

- Blender (to model, rig and adjust all the 3D models used in the project)
- Adobe Photoshop (to edit, adjust and create all the textures in the project)
- Adobe Illustrator (to create all the images and icons of the website)
- FLStudio
- Glyphr Studio (online tool at https://www.glyphrstudio.com/, to edit the “Arial” OTF font file for the 3D text geometries and add additional glyphs such as the Crab and music melody symbols of the 3D scenes’ keyboard keys)
- Facetype.js (online tool at https://gero3.github.io/facetype.js/, to generate the typeface used to create the text geometries mentioned above)

External Resources

External 3D resources:

- Crab model downloaded from Sketchfab (not rigged)
  - Sketchfab link: https://sketchfab.com/3d-models/crab-40c62bb210eb4b14a3def84461becd12
  - The model was rigged from scratch using Blender, producing a hierarchical model which was animated in JavaScript using both THREE.js and tween.js animation functions
  - The main textures of the model were reworked by me using Photoshop to fit the style and theme of the project, as well as to change the crab color for the various scenes versions

    - The geometry of the model was also simplified using Blender for performance optimizations
- Bass instrument model created by me
    - Blender was used to model the bass and to texture it using Blender’s built-in UV Texture paint
    - The texture was also color-adjusted using photoshop to fit the style and theme of the project
- DJ station instrument model created by me
    - Blender was used to model the bass and to assign simple UV texture coordinates to the various model faces using a color palette texture created using Photoshop
- Guitar instrument model downloaded from Sketchfab
    - Sketchfab link: https://sketchfab.com/3d-models/cartoon-guitar-0c8c8270cc584e94bd1fd04370bcb4dd
    - The guitar model was significantly reworked for the project: its geometry was highly simplified and changed to fit the theme and main crab model geometry and structure
    - The textures of the model (especially the color texture) were almost completely reworked using Photoshop
- Synthesizer instrument model created by me
    - Blender was used to model the synthesizer instrument, and to also assign basic material properties to the model’s vertex groups using Blender’s shading functions
- Piano instrument model created by me
    - Blender was used to model the piano instrument, and to also assign basic material properties to the model’s vertex groups using Blender’s shading functions
- Typewriter button model created by me
    - Blender was used to create the simple mesh of the typewriter buttons, with no textures or material properties defined a-priori

All 3D models mentioned above (stored in the subfolders of the “/3d/…” folder) were edited in Blender and exported as GLTF files, which are in turn imported in the main 3D scenes in the “main.js” script using the THREE.js “GLTFLoader” module.
After their loading is complete, the meshes in the scene hierarchy of these 3D models are traversed and are assigned various material properties (color, shininess, emission, metalness, roughness, and/or various textures such as color texture, normal map textures, metalness textures and/or roughness textures) which depends on the selected scene’s version type (among the 3 possible choices from the main menu).

During their initialization, models are also assigned various scales, positions rotations and additional properties which are then used for their animation (e.g. crab's armature, bones and skeleton, crab's inverse kinematics for animations, instruments models animation, ecc… see later).

Other external resources:

- Images:
    - Color, roughness, metallic and normal textures for the generated 3D meshes for the THREE.js scene (read below), as well as for the imported 3D models mentioned above (all textures are stored in the subfolders of the "/3d/…" folder)
    - The main logo and button images used in the main menu (stored in the "/images" folder)
    - The icons used to show the UI in the main THREE.js scene (stored in the "/images/icons" folder)
- Sounds:
    - All sounds were created by me using FLStudio and its default plugins (to create beat loops and melody loops) and synthesizers (for sound design of other audio sources, such as crab claws and walking sounds)
    - All the sounds were then exported as "WAV" files (stored in the "/sounds" folder and its subfolders) and imported in the scene using the THREE.js "AudioLoader" module paired with the THREE.js "AudioListener", which, using the Web Audio API, was used to play audio sources and define their basic properties (e.g. volume or playback speed).

Main 3D scenes

The web application includes 3 different 3D scenes in a THREE.js environment, in which various basic elements are defined:

- A perspective camera (camera view, position, zoom and focus point can be changed using mouse controls)
- A render canvas paired with a corresponding HTML canvas element in which the scene is drawn, and also with a THREE.js renderer
    - The "render()" function is used to draw and animate the 3D scene at each frame: the scene is drawn onto a canvas which starts from a "clear color" which is animated and changes based on the camera view angle
    - The renderer is also "tone mapped" using
- Various light sources:

    - A point light which illuminates the scene and casts shadows (using a restricted “shadow camera” frustum for a better soft shadows look);
    - An hemisphere light, which allows to tune the colors of the light and shadow areas of each mesh, and which matches the color of the background to allow to simulate a more realistic “bounce light” look (this is also animated in response to camera view position, rotation and zoom changes);
    - A global “ambient light” to illuminate areas not reached by the point light;
- A ground plane mesh, to which a THREE.js “ShadowMaterial” is assigned: this allows the plane to receive shadows casted from other objects in the scene, but otherwise is completely transparent (and lets to see through the ground plane and therefore see the canvas’ background color);
- An effect plane mesh, slighly above the ground plane, to which a particular THREE.js “MeshBasicMaterial” is assigned:
    - The effect plane’s material has a color texture mapped to another HTML canvas element in the HTML document (“#effect-canvas” element), as defined in the “create_basic_objects()” function in the “main.js” script;
    - This canvas is initially blank, but when the music is started by the user, it displays an animated rectangle which, in loop, gets bigger and bigger and gradually fades out;
    - This effect is obtained by drawing the rectangle in the canvas using the “fillRect()” function from the canvas element’s rendering context, at each frame, with parameters to control the rectangle’s size, opacity, and color;
    - This is done in the defined “initialize_effect_canvas_animation(...)” and “animate_effect_canvas_single_rectangle()” (plus some other auxiliary function) functions in the “main.js” script.

Most of these basic scene elements have properties that change based on which one of the 3 scenes versions is selected from the main menu.

Various event listeners are also assigned to the scene, at its lad to allow for the various user interactions (click of mouse with raycasts to determine the clicked geometries, keyboard button presses linked to 3D scene keyboard buttons and their animations, mouse scroll and camera movement events to change camera view, ecc…).

Procedurally Generated Meshes

The 3D geometries present in the scene and not mentioned in the list of external 3D resources above were procedurally generated using THREE.js basic shapes geometry functions (boxes and cylinders).

These geometries were then linked to materials defined using THREE.js material functions, by creating a 3D mesh to add to the scene (at the start of the scene itself). The most important 3D mesh created using this method is the main keyboard geometry for all the versions of the website (with the exception of the “typewriter keyboard version”’s buttons, which were imported as GLTF models as mentioned above).
The main 3D keyboard is in fact procedurally created in the “create_keyboard(...)” function from basic THREE.js box geometries (with various materials, normal and metalness textures assigned based on the version of the synthesizer) to constitute the base of the keyboard, the 5 crab bases and the other decorations added based on the specific keyboard version chosen from the main menu.
The buttons of the keyboard are created looping through a list of strings containing each key to display with a 3D button:

- For each letter’s key, the geometry of the buttons for both the “PC” and “Laptop” 3D keyboard in the scene are created using the “create_keyboard_button(...)” function:
    - This function starts from an array of vertices (three.js Vector3 objects containing the x, y and z coordinates of each vertex) and an array of faces (triplets of indexes of the vertices stored in the aforementioned vertices array which compose the triangle faces of the final geometry);
    - The coordinates of each vertex of the button’s geometry is then procedurally defined to allow to adjust the width, length and height of the buttons separately (e.g. to create both high buttons used for the “PC keyboard” version and lower buttons used for the “laptop keyboard” version, as well as a wider button used for the spacebar’s 3D geometry).
    - The arrays of vertices’ coordinates and faces’ vertex indexes are then combined into a “BufferGeometry” which allows to generate a 3D geometry from the defined vertices (WebGL positions) and triangle faces (the indexes of the array of positions).
    - For the “typewriter keyboard” version, this function loads a GLTF model of the typewriter button itself (mentioned above) and returns only its geometry (not its entire mesh), to then build the complete button mesh (and thus assigning the corresponding materials) in the same way as for the other 3D keyboard versions of the 3 main scenes
- For each letter’s key then, a “TextGeometry” is created starting from the typeface JSON object (in the “/fonts” folder) to create the letter to display in the button (or the symbol in the case of the “musical note” and “crab” symbols, which were embedded into the typeface JSON file starting from SVG icon files created using Adobe Illustrators, and in turn assigned to 2 special characters of the “Arial” OTF font file, in the “/fonts” folder too, using the “Glyphr Studio” online font creation tool)

- For each button then also decorative elements are created (e.g. the button base which glows when the button is held down).

Some versions of the 3 different keyboards also contain decorative elements, again assembled using the “create_keyboard(...)” function using only basic THREE.js geometries.
The keyboard and keyboard buttons were also assigned materials (MeshStandardMaterial) with different colors, roughness, metalness, and emissive colors based on the corresponding keyboard/scene style or version chosen.
Materials are oftentimes also assigned normal and metalness maps (textures loaded though THREE.js texture loaders, stored in the “/3d/general” folder): the “typewriter keyboard” version include the metalness map assigned to the 5 colored squares bases and the main keyboard, while the “laptop keyboard” version uses a normal map for the keyboard base.

Imported Meshes

The main model used in all of the 3 scenes of the application is the crab 3D model.
After the model is loaded using the GLTFLoader load function, a callback function is executed, assigning to the crab various material properties (its part’s meshes uses a “MeshPhysicalMaterial”), which depend on the keyboard scene version chosen from the main menu (e.g. 3 different color textures are assigned to the crab for the 3 versions).
Various variables later used for the crab animations are also set (e.g. storing the original positions of each of the crab’s skeleton bones, storing the original crab position, rotation and scale, ecc…).
As mentioned above, the imported crab model didn’t have a skeleton.
It was rigged from scratch using Blender, producing the following hierarchical bones structure:

```
0: root (parent: Armature)
1:    body_back (parent: root)
2:       leg_front_1L (parent: body_back)
3:          leg_front_2L (parent: leg_front_1L)
4:             leg_front_3L (parent: leg_front_2L)
5:                leg_front_effectorL (parent: leg_front_3L)
6:       leg_mid_front_1L (parent: body_back)
7:          leg_mid_front_2L (parent: leg_mid_front_1L)
8:             leg_mid_front_3L (parent: leg_mid_front_2L)
9:                leg_mid_front_effectorL (parent: leg_mid_front_3L)
10:      leg_mid_back_1L (parent: body_back)
11:         leg_mid_back_2L (parent: leg_mid_back_1L)
```

```
12:             leg_mid_back_3L (parent: leg_mid_back_2L)
13:                 leg_mid_back_effectorL (parent: leg_mid_back_3L)
14:     leg_back_1L (parent: body_back)
15:         leg_back_2L (parent: leg_back_1L)
16:             leg_back_3L (parent: leg_back_2L)
17:                 leg_back_effectorL (parent: leg_back_3L)
18:     arm_1L (parent: body_back)
19:         arm_2L (parent: arm_1L)
20:             arm_3L (parent: arm_2L)
21:                 arm_4L (parent: arm_3L)
22:                     arm_claw_topL (parent: arm_4L)
23:                     arm_claw_bottomL (parent: arm_4L)
24:                     arm_effectorL (parent: arm_4L)
25:     arm_targetL (parent: body_back)
26:     body_front (parent: body_back)
27:     leg_front_1R (parent: body_back)
28:         leg_front_2R (parent: leg_front_1R)
29:             leg_front_3R (parent: leg_front_2R)
30:                 leg_front_effectorR (parent: leg_front_3R)
31:     leg_mid_front_1R (parent: body_back)
32:         leg_mid_front_2R (parent: leg_mid_front_1R)
33:             leg_mid_front_3R (parent: leg_mid_front_2R)
34:                 leg_mid_front_effectorR (parent: leg_mid_front_3R)
35:     leg_mid_back_1R (parent: body_back)
36:         leg_mid_back_2R (parent: leg_mid_back_1R)
37:             leg_mid_back_3R (parent: leg_mid_back_2R)
38:                 leg_mid_back_effectorR (parent: leg_mid_back_3R)
39:     leg_back_1R (parent: body_back)
40:         leg_back_2R (parent: leg_back_1R)
41:             leg_back_3R (parent: leg_back_2R)
42:                 leg_back_effectorR (parent: leg_back_3R)
43:     arm_1R (parent: body_back)
44:         arm_2R (parent: arm_1R)
45:             arm_3R (parent: arm_2R)
46:                 arm_4R (parent: arm_3R)
47:                     arm_claw_topR (parent: arm_4R)
48:                     arm_claw_bottomR (parent: arm_4R)
49:                     arm_effectorR (parent: arm_4R)
50:     arm_targetR (parent: body_back)
51: leg_front_targetL (parent: root)
```

52:	leg_mid_front_targetL (parent: root)
53:	leg_mid_back_targetL (parent: root)
54:	leg_back_targetL (parent: root)
55:	leg_front_targetR (parent: root)
56:	leg_mid_front_targetR (parent: root)
57:	leg_mid_back_targetR (parent: root)
58:	leg_back_targetR (parent: root)

The “target” and “effector” bones in the hierarchical structure are used for the crab’s inverse kinematics.
After its loading, in fact, the crab model’s “armature” skeleton is used to define an inverse kinematics functionality for all of the 8 crab legs and for the 2 crab arms.
This is done by setting an array of javascript objects defined as follows:

```
{
        target: <target_index>,        // The index of the bone representing the IK target
        effector: <effector_index>,    // The index of the bone representing the IK effector
        links: [                       // An array of objects containing the indexes of the
                {                         bones affected by this target/effector's IK, in
                        index: 4,         hierarchical order
                        rotationMin: <angle>,
                        rotationMax: <angle>
                },
                {
                        index: 3,
                        rotationMin: <angle>,
                        rotationMax: <angle>
                },
                {
                        index: 2,
                        rotationMin: <angle>,
                        rotationMax: <angle>
                }
        ],
        iteration: N                  // The number of iterations of the defined IK algorithm
}                                        to perform at each IK solver's update (each frame)
```

The array of object thus defined also includes a max and min rotation angles for the various joints in the “links” array.
Each of these object defines an inverse kinematics joints list with an associated target and effector, i.e. the destination position in which the effector bone needs to be.

The inverse kinematics functionalities for the crab's joints use the Cyclic Coordinate Descent algorithm (CCD) to solve joint configurations and therefore calculate the joint angles to match the positions of the effector and the target, with 3 CCD algorithm iterations per frame (tradeoff between precision and performance).
The inverse kinematics of the crab are used for his animations, mainly the walk animation:

- The main bone of the crab is moved in the clicked position onto one of the crab bases (using a tween.js animation) while the targets of the legs lag behind;
- After the main bone reaches a certain distance from a leg's IK target, the target is moved to said position (again, using a tween.js animation which simulates a leg movement)
- If a target for a leg needs to move but the opposite leg is not grounded, the target is not moved: this creates an organic and realistic enough walk animation for the crab.

The crab's "dance" animation (or "idle" animation) also exploits the inverse kinematics of the leg, by keeping the crab leg's anchored to the ground while the main body's bone moves up, down, left and right.
Inverse kinematics is also used for the crab arms in some phases of the crab animations for the various instruments.

Models for each of the 5 instruments (corresponding to the 5 crab bases) are imported in the same way as the crab model, by defining their basic material's properties and variables later used for animations.

Animations

Various animations are used in the scene (e.g. color changes animations for keyboard buttons, keyboard button presses based on user's computer keyboard presses and interaction, color changes for the background and meshes in the scenes, camera view animations, ecc…, but also the effect canvas animation for the ground plane, or other more complex animations).

The main animations are the crab animations, and its interaction with the various instruments:

- Crab walk (mentioned above): the crab moves where the user clicks and stands there (alo includes other sub-movements, such as the up and down movement of the crab main bone to mimic real world crab movements);

- Crab dance: while any sound is being played, the crab dances by moving his main bone up, down, left and right to the beat (this animation is also blended together with the various instruments interaction animations)
- Crab playing the 5 intruments:
    - These animations consists of first moving the crab bones and each of the instruments to a certain start position and rotation (using a tween animation), and also animating the scale, colors, and other relevant properties of both the crab and each the instrument to play.
    - Then the actual animation is started and looped, which involves the instruments pulsing, glowing, changing their scale, position, rotation, ecc…, and also involves the crab's arms interacting with the instruments

Each of these animation is generated at scene load using tween.js tweening functions and THREE.js functions for color / material properties changes and also for position, rotation and scale changes.

Sounds System

The various sounds are loaded at scene start (as mentioned above), and played and looped using the "sounds_looper" function when the user interacts with the scene elements (e.g. presses of keyboard or click of crab bases), based on a certain beats per minute tempo.

A "KeyboardButton" class is used to store objects representing each of the keys available and their assigned sounds, and various properties which define the ebay the various sounds need to be played (e.g. melody sounds, kick-snare sounds, hi-hats sounds, sub-bass sounds, ecc…).