Interactive Graphic Project

Matteo Romano 1783252
romano.1783252@studenti.uniroma1.it

1 Introduction

The main topic of my work is a game where we have an agent moving in a map and several other agent as enemy that we have to fight. I worked mainly to a suitable AI for the agent with increasing complexity respect the game's difficulty. Other important element that I wanted to higlight are the animation. Several of theme are created this because an increase in the number of agent's action meaed also an increased complexity of the level design of the game with concious limits due to fps that can occasionally increase and decrease but always more than 20 depending on the number of models loaded in the envoirnment and their complexity.

2 Project Requirenment

1. Tools and Envoirnment: Here i discuss The exaustive list of all envoirnment and libraries used in this project:

- **WebGL:** To manually create assets for the envoirnment mainly used for the shading over bulbLight from ThreeJS, I noticed that even well structured model required a fine tune of the textures.
- **Blender:** Used to modify the more tricky effects of the armatures or character's bodies and modify textures and models imported for the envoirnment (ususally merged with webGL created assets).
- **TweenJS:** Used for the interpolation of the spline created for the smoothed animation of **Sect. 3.1** creating a function to send for the interpolation for each class. This is done using the $Tween\_group$ library respect a key-value for each joint to move as reported in the set of animations to move the character.

2. Animation: I implemented in **Sect. 3** the animation translated from degree to quaternion as required for the model applied to the hiretical structure fine tuned by me using $Get\_rotation\_quaternion.ipynb$. The list of animations is:

- **Attack:** static position for legs limbs but with moving blade.
- **Defense:** static position for all limbs but with fight pose.
- **Run:** move of the entire body expecially limb
- **Rest:** simple rest pose where everithing is static no move for the limbs or hiretical stucture.
- **Rest in Combat:** Your window to attack the enemy changable in duration with the difficulty button

3. User Interaction: The user cannot change the visual you can abilitate it with orbitControl easly but you have to live your hand from the attack-defense comand so I removed it Inside this project the user can interact with:

- **Attack:** After approaching an enemy you can spam "A" comand to perform a hand-smothed animation **Sect. 3.1** to decrasce the enemy life-bar.
- **Defense:** After approaching an enemy you can use "B" command to perform a hand-smothed animation **Sect. 3.1** to avoid decreasing of your own life points.
- **Run:** Use the directional arrows to move the character.

- **Heals:** Using D you can heal yourself in specific regions wher eyou have. amagic grass but only whene the timer to do it is "READY" and paying 3 kill points.

- **Dashboard:** With this tool provided by threeJS I add the interaction on **Level design** as difficulty, **Enviornment light** with a day-night switch with a torch as illumination and the reset button.

4. Model: The model of the characters is just taken from mixamo and modified adding texture shadings with high roughness to simulate metals material over dynamical lights.

3 Envoirnment

For the various asset I used a combination of blender models created by me using pre-built models and WebGL to create rocks and walls for the envoirnment. First I put procedurally generated rocks starting from a $SphereGeometry$ with noise in the positioning and the values of poligones for the mesh creation. This means that each rock is in a group with number of random elements between 0 and 5 with different shapes. I created at least thirty rocks and as maxymum (random) 30*5=150 rocks of different shapes. All of them are threated with specific material values **Tab 1.** for light and shading. The rocks as several (not all) assets makes use of the functions as: $box.setFromObject(boxMesh, true)$ for the hit-box computation **Sudhgestion:** use them to hide from enemy when your healt is low or they has been actracted more than one at time anyway they are threated from the enemy's AI they will loose you after passing around theme. At the extreme of the map is created a ring similarly with a boxGeometry. The grass instead is threated with a group of meshes from 1000 to 100000 the number has been drastically decreased due to the low level FPS reached. The noise for the direction is managed chosing randomly position and deflection of the mesh but all with respect the same normal direction to simulate wind over the grass **Fig. 4**. The grass meshes are defined with a fragment shading to illuminate for light and choosed a vec2 vector as main color. The magic grass will have also a use in terms of game design considering the possibility to heal the player only in their position with a countdown every n random seconds and after having killed at least 3 enemies. Their color much bright is better to better see far away elements even with fog. The same is done for random element added to the scene to improve the apparance of the world as statue and houses.

Figure 1: Exemple of simpre rock's meshes with texture applied they are in groups usually and with different shapes.

3.1 Illumination

You can check in the next picture the results in a more visible situation (with wood texture for the floor) the effect on the mesh of a dynamic illumination and the resulting shadow performed for object and models. ThreeJS provide both illumination of the envoirnment and the Sphere of light respectevly using the PointLight and HemisphereLight. In my case I used just one of theme for the night to simulate the torch effect over the body armor and the envoirnment. While the HemisphereLight is usd in combination with other four PointLight on the top of the user's model to better show the armor shiness. To attach a pointlight over the position of the torch of the model I used the position.set comand respect the model position summed with a distance multiplied the sine(cosine for z-coordinates) of the rotation angle of the model respect the sceene. To perform light is used an if else codition checking for previous lights shadow maps and in case a new one is present changed with ballMat.needsUpdate = true; cubeMat.needsUpdate = true;floorMat.needsUpdate = true;previousShadowMap = params.shadows.

3.2 Floor

For The construction of the hill similarly to the example of the kangaroo, I just used a meshing algorithm to create the various polygons to elevate the ground according to a certain function. The function choosed is similar to a three dimensional **Gaussian** function with cuts over x,y and z accord-

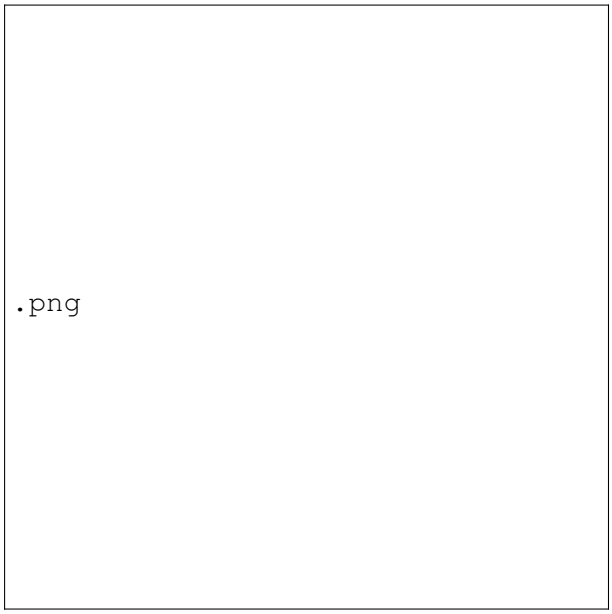

Figure 2: In the picture you can se a preliminar test where pre built models are used with a shadow coming from a wall near to theme it is choosed this photo because easly visible respect rock irregoral shadows.

ing to the end of the hill and start of flatten area. Anyway Has been quite complex to do a depression like in real hills exluding mesh by mesh the useful ones for the function of a sphere. So I created for such problem the hill with a different function respect the one of the ground and cut in such a way to merge this with the ground properly creating a smouthed depression. The function that perfectly do this is the "hat function":

$$f(r) = \frac{\sin \pi r}{\pi r} \tag{1}$$

With $r = \sqrt{x^2 + y^2}$ cutted (during the pushing of meshes's verticies) for the x and z (plane of the ground) and squeezed with a scaling matrix to obtain a "wide hat". For such problem has been choosed the texture's normal respect the normal of the direction of each mesh. To define the various (x_i, y_i, z_i) tuples to create the verticies of the mashes I just used the **(1)** to define the elevation of the hill (y_i) and pushed it with $x = 2{*}i/nRows - 1$ and $z = 2 * j/nColumns - 1$ (result in **Fig. 2**).

4 Hiretical Structure

The original model came from mixamo webpage and is visible at https://threejs.org/examples/#webgl_animation_skinning_additive_blending. The final result taken as it is with the following hiretical structure in **Fig.** is taken and combinated with another model of an armor and added then to the model. On it several manipulation of the texture has been performed using blender as roughness variation over texture or respondance on illumination adding a test light (removed before the export to the Js file). The final resulting bones will be achived only by my program using the simple getObjectByName procedure and then use the corresponding Object3D obtained for the manipulation of position through quaternions after an initial rest static configuration. In the code the model is loaded asyncrhronously with other assets at the start and a rop of Fps is obtained by this for this reason I preferred to avoid too much elements in the sceene.

5 Characters Behaviours

I defined the position of the models positioned over the z-x axis of the $PlaneGeometry(m, n)$; plane. The same is done respect the enemy and at each time step, using the system call **Date.now()**, and getting the current position of the model at a certain time-step. Using $x = r \sin \theta$ and $y = r \sin \theta$ variating only the θ value and setting the torso direction respect the normal over the circle just using $\theta^{/} = \frac{\pi}{2} - \theta$ all the model can move inside the envoirnment just with θ, ρ values. I devided this section in Animation and behaviours to differenciate the local movment of models respect the envoirnment evolution and the behaviour respect the game design requirements.

5.1 Enemy patterns and combat system

The resulting behaviour of the enemy is in this case based on a initial state of relaxed position of the model exept for the hand on the side of the model. Everithing is managed by a set of dictionaries one for each aspecto of the possible states an agent can be in the grid world **Fig. 2**. Each agent can be in the following states:

- **Searching for opponents:** The initial behaviour is search the player to kill just a walk in random direction scanning the areas with a window o view of fiew meters.
- **sighted enemy:** In this case we have seen him we have to approach but we can't still fight he(the user) could decide to run away and he is faster.
- **Fight:** In this case the ser is in area to be attacked and will start a random choice between

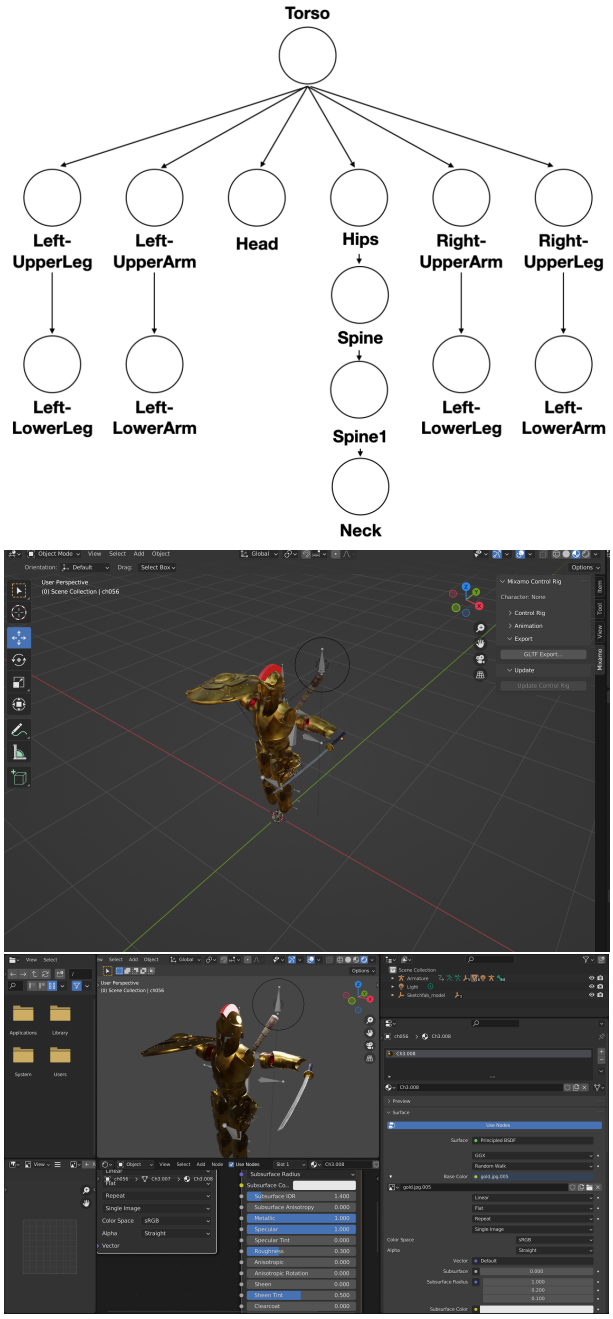

Figure 3: This is the hiretical structure where the foot, hand and eyes are omitted of the structure to use for the rotation's translation/rotation. The final result of the rotation is respect each reference frame without considering rotation respect the world frame. For the values of legs and spines are related other bones/meshes attached for armature.The work performed in the second picture has been respect the construction of the attached bones and then the connection with the armor. A post moothing procedure has been applyied to the greave for fit the legs correctly and it is done manually. The modeling included models of blade, shield and torch. The last image has been instead respect the creation of the light effects. You can see a light over the torched to test it. The roughness over a gold texture is decreesed drastically to obtain that output.

attack, defence and take a breath to give opening window for the player.

- **Escape:** The user for some reason run away and will be followed by the enemy until the distance between user and enemy will be enough far in that case the user is escaped and everithing returen in the first case condition.
- **Killed:** The enemy life goes under a certain value and it simply respawn randomly far away from the user giving 1 point to the kill counter.
- **User Killed:** The User life goes under a certain value and the game end setting again all the flags to their original value.

In order to properly perform the fight the rotation of the enemy must full fill the direction of the opponent. To do so Has been enough to define the angular coefficient of the line passing througth the two points and using the usual formula:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

Calculating the atan(m) function of it and checking the values of x and y of each point in the four quadrants I obtained the direction and angle for the fight and trajectory following for the enemy to fight the human.

5.2 Animations

For the animation part I used at beginnong blender and tried to export the resulting rotation manually from its dashboard to an array of dictionary one for each animation. Then I used for many joints an hand made program to generate a ArrayFloat of n different degrees using splines and translated with another .ipynb program from degree to quaternions to pass to the hiretical structures enemies and main character. The value of quaternions calculated are respect the sequence of rotations obtained by the teenJS values creating the rotations sequences from euler angles to tween input sequences of quaternion and at the end obtaining the finale interpolated sequence from TweenJS. To move properly the bodies parts with respect each position of the entire animation I had to just use rotation properly for each body part. Having a simple movement by translation of legs position will not create smooth trajectory. So I used spline to define both the ,attack body parts rotations and walk time steps in order to interpolate

```
choose_direction = {'enemy_1': 0,'enemy_2': 0,'enemy_3': 0};
choose_distance = {'enemy_1': 0,'enemy_2': 0,'enemy_3': 0};
reached_objective={'enemy_1': true,'enemy_2': true,'enemy_3': true};
objective_x = {'enemy_1': 0,'enemy_2': 0,'enemy_3': 0};
objective_z = {'enemy_1': 0,'enemy_2': 0,'enemy_3': 0};
timer_enemy = {'enemy_1':  Date.now(),'enemy_2':  Date.now(),'enemy_3':  Date.now()};
enemy_spotted = {'enemy_1': false,'enemy_2': false,'enemy_3': false};
flag_enemy = {'enemy_1': 3,'enemy_2': 3,'enemy_3': 3};
life_enemy = {'enemy_1': 4,'enemy_2': 4,'enemy_3': 4};
////
else if(!((MyCharacter.model.position.x < visual_enemy_xp) && (MyCharacter.model.position.x > visual_enemy_xm) &&
        (MyCharacter.model.position.z  < visual_enemy_zp) && (MyCharacter.model.position.z > visual_enemy_zm))
        && enemy_spotted[name_enemy]){
            enemy_spotted[name_enemy]=false
            reached_objective[name_enemy]=true
        }

////////////////// arrive to the objective ///////////
else if(enemy_spotted[name_enemy] && reached_objective[name_enemy]){

    enemy_in_combat[name_enemy]=true
    in_combat=true

// FIGHT BEGIN
    var coeff_ang = (t1.model.position.x-MyCharacter.model.position.x)/(t1.model.position.z-MyCharacter.model.position.z)
    if(t1.model.position.x-MyCharacter.model.position.x>0 && t1.model.position.z-MyCharacter.model.position.z>0){
    choose_direction[name_enemy] = Math.atan(coeff_ang)+Math.PI
    }
    else if(MyCharacter.model.position.x-t1.model.position.x>0 && MyCharacter.model.position.z-t1.model.position.z>0){
    choose_direction[name_enemy] = Math.atan(coeff_ang)
    }
    else if(t1.model.position.x-MyCharacter.model.position.x>0 && MyCharacter.model.position.z-t1.model.position.z>0){
    choose_direction[name_enemy] = Math.atan(coeff_ang)
    }
    else if(MyCharacter.model.position.x-t1.model.position.x>0 && t1.model.position.z-MyCharacter.model.position.z>0){
    choose_direction[name_enemy] = Math.atan(coeff_ang)+Math.PI
    }
    if(Date.now()-timer_enemy[name_enemy]>2000 && flag_enemy[name_enemy]!=1){
        var my_flag = 1 + Math.floor(Math.random() * (4 - 1 + 1));
        if(my_flag==3){
            my_flag-=1
        }
        //console.log(my_flag)
        flag_enemy[name_enemy]=my_flag
        timer_enemy[name_enemy]=Date.now()
    }
    else if(Date.now()-timer_enemy[name_enemy]>evade_window && flag_enemy[name_enemy]==1){

        var my_flag = 1 + Math.floor(Math.random() * (4 - 1 + 1));
        if(my_flag==3){
            my_flag-=1
        }

        //console.log(my_flag)
        flag_enemy[name_enemy]=my_flag
        timer_enemy[name_enemy]=Date.now()
    }

    if(Math.abs(MyCharacter.model.position.x-t1.model.position.x)>1.5 || Math.abs(MyCharacter.model.position.z-t1.model.posit

    enemy_in_combat[name_enemy]=false
    in_combat=false
        flag_enemy[name_enemy]=3
        reached_objective[name_enemy]=false
        enemy_spotted[name_enemy]=true
```

Figure 4: The picture shows the main flags used for the state creations of the agents. While the second picture is the code ofthe fight states as example of possible states where we can be and it's corresponding behaviour.

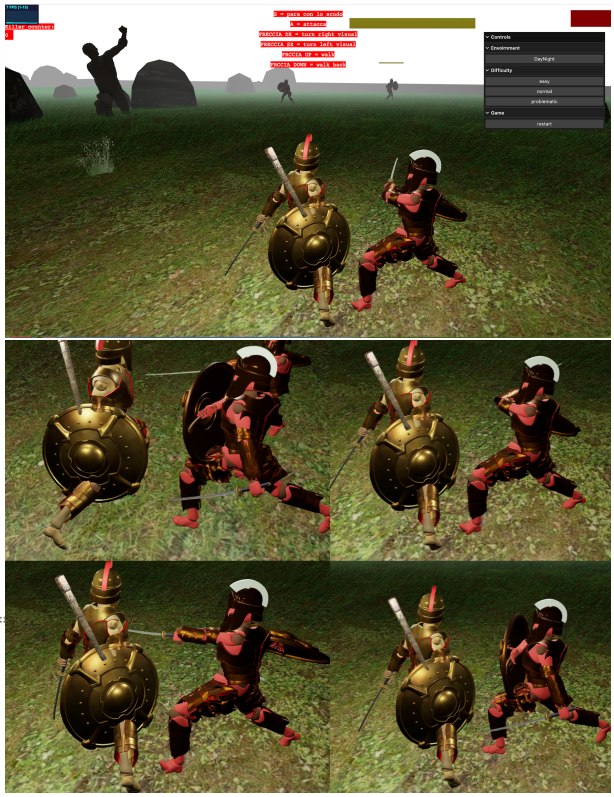

Figure 5: In the figure 1 you have a complete view of the wotk with fps counter as usual very low, life of enemy as sprite and other aspect related the game livness. Possible animations viewed in the fight with even more than one enemy. The position of the hips must change in this case and with it the position of the camera to full fill the scene of the fight.

each position of the key-frame animation. To do so I defined a third order polynomial over a time interval of 2 seconds. with T=1 and $\tau = \frac{t}{T} \in [0, 1]$ the function is in the form:

$$q(\tau) = q_{in} + \Delta q(a_0\tau^3 + a_1\tau^2 + a_2\tau + a_3) \quad (3)$$

and computed the coefficient (a_0, a_1, a_2, a_3) interpolating a set of position given by the joints rotation of each link. The value of τ is calculated at each iteration of the animation as $currentTimeStep - \max Valueinterval$. Important tip for this calculation is that the trajectory computation is quite easy in this case due to a rest to rest trajectory defined for the spline $(v_s = v_f = 0)$ so with a rest-to-rest computation of the coefficients. To compute it I used the formula of spline with A:

$$\begin{pmatrix} 2(\Delta h_1 + \Delta h_2) & h_1 & \dots & \dots & \dots \\ h_3 & 2(\Delta h_2 + \Delta h_3) & h_2 & \dots & \dots \\ h_4 & h_5 & 2(\Delta h_3 + \Delta h_4) & h_3 & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & h_{n-1} & (\Delta h_{n-3} + \Delta h_{n-4}) & h_{n-3} \end{pmatrix} \quad (4)$$

And the b known vector matrix as:

$$\begin{pmatrix} \frac{2}{h_1 h_2}(h_1^2(q_3 - q_2) + h_2^2(q_2 - q_1)) - h_2 x_1 \\ \frac{2}{h_2 h_3}(h_2^2(q_4 - q_3) + h_3^2(q_3 - q_2)) \\ \dots \\ \dots \\ \frac{2}{h_{n-2} h_{n-3}}(h_{n-3}^2(q_{n-1} - q_{n-2}) + h_{n-2}^2(q_{n-2} - q_{n-3})) \end{pmatrix} \quad (5)$$

Then we calculated the unknown using the inverse of the matrix the A as $(x_0, x_1, x_2)^T = A^{-1}b$ and we used the result (starting from 0 and with last state 0) to find the first coefficient a_1 and the original points for a_0. After this we computed the a_2, a_3 with the matricies:

$$A' = \begin{pmatrix} h_k^2 & h_k^3 \\ 2h_k & 3h_k^2 \end{pmatrix} \quad (6)$$

and the coefficient that are known as:

$$b = \begin{pmatrix} q_{k+1} - q_k - x_k h_k \\ x_{k+1} - x_k \end{pmatrix} \quad (7)$$

The final result given will be than translated to a quaternion rotation sequence to full fill the requirement of the quaternion with the formula on quaternion translation considering the various rotation matrix respect each axis and translating each element of quaternion with:

$$\textbf{x-axis} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) \\ 0 & sin(\text{sin}) & cos(\gamma) \end{pmatrix} \quad (8)$$

$$\textbf{y-axis} \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \quad (9)$$

$$\textbf{z-axis} \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (10)$$

Then I computed for each quaternion component a final value to pass for the spline over the computation for each joint.

$$q_0 = cos(\gamma/2)cos(\beta/2)cos(/2) + sin(\gamma/2)sin(\beta/2)sin(/2) \quad (11)$$

$$q_1 = sin(\gamma/2)cos(\beta/2)cos(/2) + cos(\gamma/2)cos(\beta/2)sin(/2) \quad (12)$$

$$q_2 = cos(\gamma/2)sin(\beta/2)cos(/2) + sin(\gamma/2)cos(\beta/2)sin(/2) \quad (13)$$

$$q_3 = cos(\gamma/2)cos(\beta/2)sin(/2) + sin(\gamma/2)sin(\beta/2)cos(/2) \quad (14)$$

And at the enf I created a spline function with this parameters and for each value between 0 and 2 seconds for the duration. The animation duration is also considered respect the game difficulty for example the enemy have a rest-in-fight animation of 3 seconds to 2.5 depending from difficulty.

6 Audio

The audio present in the game is related only to sound effects for attack animations and defense and from music that I used as background noise. With soundtrack as: mixkit-metal-bowl-hit-1842.wav. mixkit-metal-medieval-construction-818.wav. mixkit-metallic-sword-strike-2160.wav. mixkit-quick-knife-slice-cutting-2152.mp3. and as music the $sb\_indreams.mp3$.
Is maneaged directly from the states where the enemies is because is the only case whene these sound can be produced like in **Fig.4**.

```
            ////////////////
if(flag_sound["attack"]==1){
            sound.stop();
            sound_attack.stop();
            flag_sound["attack"]=0
            stop_sound["attack"]=Date.now()
        }

if(flag_sound["defense"]==1){
            sound_shield.stop();
            //sound_attack.stop();
            flag_sound["defense"]=0
            stop_sound["defense"]=Date.now()
        }
    }
```

Figure 6: Deactivation flag of sounds.

Figure 7: In order the sprite for starting and ending of the game with invisible button in Html and Css style overwritten with the blood track. The result is visible in the function of the third image. In the code above instead is reported the settings after retrying.

7 Sprites

Are used several sprites for menus and in general the ashboard necesessary to play. They are choosed because of their semplicity over the object3d elements that can be affected by illumination condition or envoirement interaction as enemies. These images can be seen in the **Fig. 8** with ending and starting images.

8 Other Libreries

- GLTFLoader.js
- FBXLoader.js
- GUI.js
- Stats.js
- tween.esm.js

- Utils.js

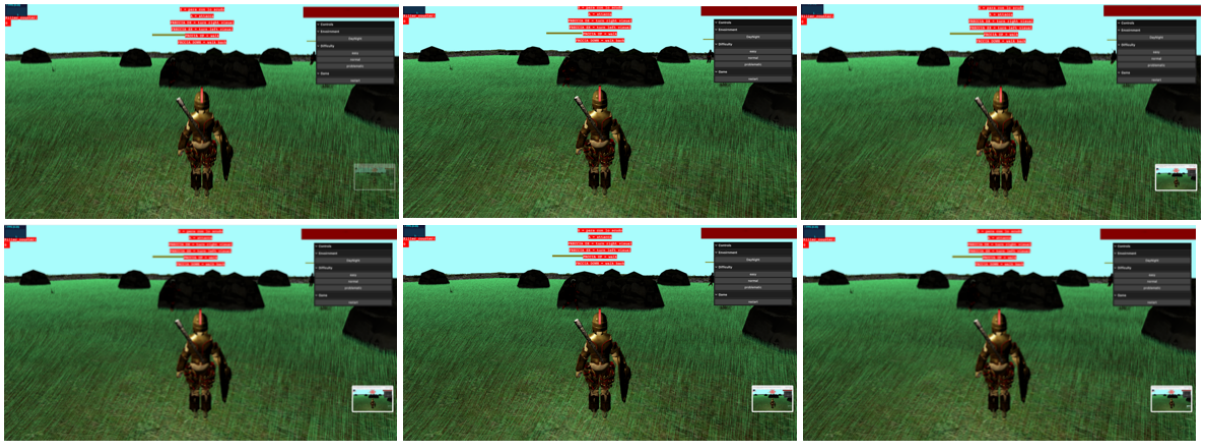

Figure 8: The picture from left up to down right shows the evolution in time of the movment of the grass in exactly 6 frame animation rendered, unfortunately to mantain acceptable levels of fps has been necesessary to reduce the number of meshes for grass from 10000 to 1000 (to see you can change theme).

```
        let simpleNoise = `
float N (vec2 st) {
    return fract( sin( dot( st.xy, vec2(12.9898,78.233 ) ) ) *  43758.5453123);
}

float smoothNoise( vec2 ip ){
    vec2 lv = fract( ip );
  vec2 id = floor( ip );

  lv = lv * lv * ( 3. - 2. * lv );

  float bl = N( id );
  float br = N( id + vec2( 1, 0 ));
  float b = mix( bl, br, lv.x );

  float tl = N( id + vec2( 0, 1 ));
  float tr = N( id + vec2( 1, 1 ));
  float t = mix( tl, tr, lv.x );

  return mix( b, t, lv.y );
}
`;
const instanceNumber = 1000000;
const dummy = new THREE.Object3D();
const geom = new THREE.PlaneGeometry( 0.001, 0.5, 1, 4 );
const instancedMesh = new THREE.InstancedMesh( geom, leavesMaterial, instanceNumber );
var rotationChoice=0
for ( let i=0 ; i<instanceNumber ; i++ ) {
    rotationChoice=0+ Math.random() * (6.28  + 1 )
    dummy.position.set(
    -20*Math.sin(rotationChoice) + Math.random() * ( 20*Math.sin(rotationChoice)  +20*Math.sin(rotationChoice)  + 1 ) ,
    0,
    -20*Math.sin(rotationChoice) + Math.random() * ( 20*Math.cos(rotationChoice)  +20*Math.cos(rotationChoice) + 1 )
  );

  dummy.scale.setScalar( 0.5 + Math.random() * 0.5 );

  dummy.rotation.y = Math.random() * Math.PI;

  dummy.updateMatrix();
  instancedMesh.setMatrixAt( i, dummy.matrix );
}
```

Figure 9: Code for the creation of the grass and noise introduction as mix of the value with the smoothNoise value obtained a value of random displacement for the wind.