Final Project Giuliano Bruzzese student Id. 1757334

ABSTRACT

This report describes design and implementation of a three-dimensional SnakeGame that works in the web browsers without the need to install any plugin, addon, or extension. The implementation is accomplished through JavaScript language with WebGL (Webbased Graphics Library) extension, that serves as binding between JavaScript itself and GPU, and is available in most of the modern web browsers.

Some of the key aspects of this work were the implementation of the algorithms that guarantee the animation of the snake, according to the classic rules of the game, and the transposition in a three-dimensional environment of a game distributed mainly in two-dimensional format. The software presented here is the result of the development of a hierarchical model, for whose nodes it was necessary to create appropriate functions and textures. The lighting present in the game is achieved through a Phong Shading approach.
The game features increasing difficulties and the ability to change the game scenario to provide players with a more immersive gaming experience.
For development and implementation it was preferred not to use specific JS libraries (ThreeJS or BabylonJS) and to manually create each element, trying as much as possible to obtain realistic environments, not neglecting attention to buffers management and software performance.

The source code is hosted in a repository on GitHub.

Figure 1 - On the left the Nokia 5110, on the right one of the first snake game.

1. INTRODUCTION

1.1 History and rules of classical Snake Game.

The decision to develop a SnakeGame stems from the fact that, between the end of the 90s and the beginning of the new century, one of the first genre video games with which the boys interacted was a snake game. This is historically due to the game variant that came preloaded on Nokia phones in 1998 (Nokia 5110), allowing snake games to find an ever-wider audience and thus a resurgence of interest in the game. Just think that in 1998, the Finnish company recorded a turnover of 20 billion euros with a profit of 2.6 billion, and that, in 2000, the peak of its history, Nokia held a share of 30% in the global mobile phone market, nearly double that of its closest competitor, Motorola. The company operated in 140 countries and alone accounted for 4% of its country's GDP, 21% of total exports and 70% of the Helsinki Stock Exchange of the capital market.

The Snake design dates back to the arcade game Blockade, developed and released by Gremlin in 1976. Although the first known home computer version, titled Worm, was programmed in 1978 by Peter Trefonas for the TRS80 and published by CLOAD magazine. in the same year. The impact that the game has on users, despite its simplicity, made it place, in 1996, in 41st place in the ranking of the "100 best games of all time" according to the Next Generation thanks to the need for both quick reactions and of foresight.

So the simplicity of the game, especially as regards the scenarios to be designed, together with the notoriety of the same, made Snake a perfect candidate for development and implementation in a threedimensional environment, which in any case represented an interesting variant of the traditional game, through JavaScript language.

Snake is a snake that by eating what appears on the display, tipically fruit, stretches and the player earns points. He is in constant motion and must avoid bumping into obstacles or against himself, which becomes more and more difficult as his body lengthens. The player loses when the snake meets the edge of the screen, another obstacle or himself. Basically, it is a video game genre where the player maneuvers a growing line which becomes a major obstacle for himself.

The gameplay is available in two main variations:

In the first, which is often a twoplayer game, there are multiple snakes on the playing field. Each player attempts to block the other so that the opponent runs into an existing trail, left by the opposing snake, and loses.

In the second variant, a single player attempts to eat objects by meeting them with the snake's head. Each eaten object stretches the snake, so avoiding collision with the snake becomes progressively more difficult.

For this project, the second variant of the gameplay was preferred because it is the most common variant and is also the one present on the Nokia 5110.

1.2 Environment used

For the software implementation it was decided not to draw on advanced JavaScript libraries, but to use only WebGL in its basic form, with the aim of achieving, in this first approach to interactive graphics, a level of realism as high as possible with their own strength, and of independently obtaining the desired effects without excessively burdening the project performances.

WebGL, as read on the Khronos Group website https://www.khronos.org/webgl/, is a crossplatform open web standard for a lowlevel 3D graphics API based on OpenGL ES, exposed to ECMAScript via the element HTML5 (HyperText Markup Language version 5) Canvas.

It allows web developers to create highperformance threedimensional presentations, which were previously only possible using browser plugins or extensions. The most significant drawback of this past approach was that the user had to have some additional software installed, such as Adobe Flash Player or Java Virtual Machine. Instead WebGL acts as a link between highlevel JavaScript and lowlevel GPU operations. It is based on OpenGL ES 2.0 (Open Graphics Library for Embedded Systems) to be easily deployable on modern mobile platforms such as Android or iOS (iPhone OS). Major browser vendors Apple (Safari), Google (Chrome), Microsoft (Edge) and Mozilla (Firefox) are members of the WebGL Working Group.

Another important aspect linked to WebGL, and which represents one of its major advantages, is that, in the past, graphics cards worked in FFP (FixedFunction Pipeline) mode and programmers didn't have much control over the rendering process, because most of the operations, i.e. geometry transformation, light and texture sampling, were built into the hardware and could not be changed programmatically. Instead, modern GPUs use the socalled Programmable Pipeline (or Shader Pipeline) which allows programmers to write special pieces of software (shaders) that describe the behavior of the pipeline, providing a very efficient way to create various effects.

WebGL is just a Shaderbased API that uses GLSL (OpenGL Shading Language). Specifically, OpenGL ES 2.0 only supports the Programmable Pipeline.

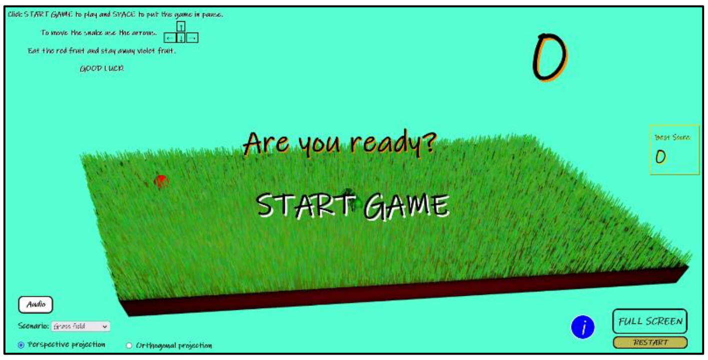
Two types of shaders are supported in the WebGL architecture: Vertex and Fragment.

Vertex shader is a program that is executed “pervertex” or, more generally, on each element of input data, vertex attributes. These data usually consist of vertex positions, vertex colors, vertex normals and texturing coordinates. Every vertex shader writes its calculation result, that is final vertex position in clipspace coordinates, to the special variable called gl_Position.

Fragment shader, instead, deals with calculating the final color of each fragment where the currently rendered object is drawn, and writes it into the FragColor variable, in windowspace coordinates. They constitute two of the fundamental steps of the rendering process in the pipeline architecture.

In the specific case of this project, the programs vertexshader and fragmentshader calculate the current position of each node of the hierarchical model and assign a color to each fragment according to the Phong shading technique, based on BlinnPhong Model.

2. TECHNICAL ASPECTS


Click START GAME to play and SPACE to put the game in pause.

To move the snake use the arrows.

Eat the red fruit and stay away violet fruit.

GOOD LUCK!

Scenario: Grass field

[✓ Perspective projection] Orthogonal projection

Figure 2 - Game homepage.

2.1 GAME HOMEPAGE

In the home of the game you can identify 5 specific sections. The first, top left, in which some game indications are briefly summarized regarding starting or pausing the game, how to move the snake in space, what to eat to grow, and then level up, and from what to stay at wide. Also in the upper part, but on the opposite side, there are two textual HTML elements, one of which acts as a scoreboard (or levelboard), and the other which serves to record the highest score obtained in the different games. The management of these two elements takes place in the Snake3D.js file through the use of the "innerHTML" function, which allows you to rewrite the contents of a text box. For each fruit eaten the "score" variable is increased by 10 points, if the "score" value exceeds that of the "bestScore" variable, then the latter is updated to the "score" value and the result of this operation is transcribed in the space provided.

In the central part of the home of the game there are the playing field and two other textual elements that are responsible for starting the game and warning the user when the game is paused.

At the bottom left there are four HTML elements: a button that turns the game audio on or off, a drop-down menu, thanks to which it is possible for the player to constantly change the game scenario, and, finally, two radio buttons, which allow you to switch from one projection to another. At the start of the game the projection is set to perspective, so the perspective radio button is checked. To manage the link between the two radio buttons it was necessary, in the Snake3D.html file, to assign them the same "name" tag.

At the bottom right, however, there are three buttons. The first takes care of opening the screen with all the information regarding the game controls. The button marked with the word FULLSCREEN allows you to switch to full screen mode. The RESTART button, accessible only at the end of a game, resets the variables to their initial value and allows you to restart a game.

2.2 BUTTONS

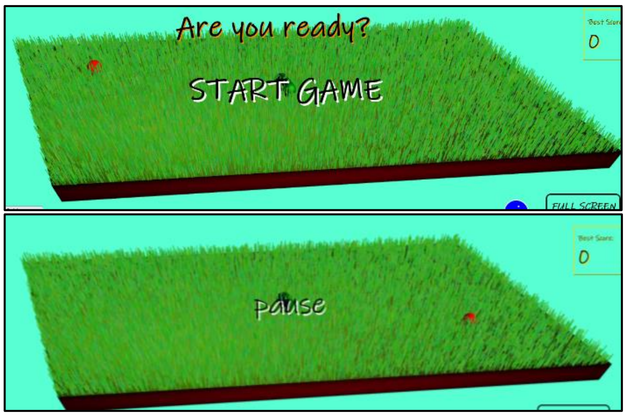


Figure 3 - Start button and pause mode.

2.2.a START BUTTON

At the beginning of the game it is possible to start the animation either by pressing the SPACE key or by clicking with the mouse on the word START GAME.

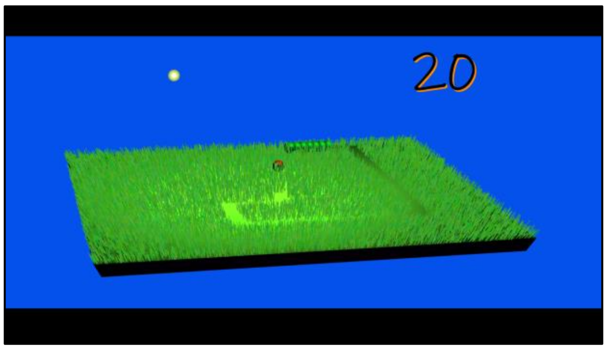


Figure 4 - "Night" scenario in full screen mode.

2.2.b FULLSCREEN BUTTON

The FULLSCREEN button allows you to resize the HTML division identified by the "FULL-SCREEN" Id. Since a canvas object cannot have children, the entire element had to be resized in order to project the score and game status in full screen as well. This unfortunately leads to a small problem when exiting full screen mode. In fact, the authorization and control that browser make on full-screen objects means that the ESC command, which is responsible for resizing the screen and repositioning the canvas on the web page, is momentarily overridden by the ESC command of specific browsers. Consequently, to correctly reposition the canvas on the web page it is necessary, once exiting the full screen mode, to press the ESC key again.

2.2.c RESTART BUTTON

The RESTART button allows you to restart the game. To manage the implementation complexity of a real restart button, not wanting to use particular cross-platform game engines, such as Unity, we opted to create a button that in fact set the snake current position values to the original ones, in this way the check on the game state (finished or not), the nature of which will be explored in the following paragraphs, will result in "not finished" and therefore give the possibility to play again. Furthermore, this button, in addition to the position, also restores other values to the initial ones, including the "score", concretely giving the user the possibility to start over the game.

Note that it is set to disabled at the start of the game and is activated only after the game is finished.

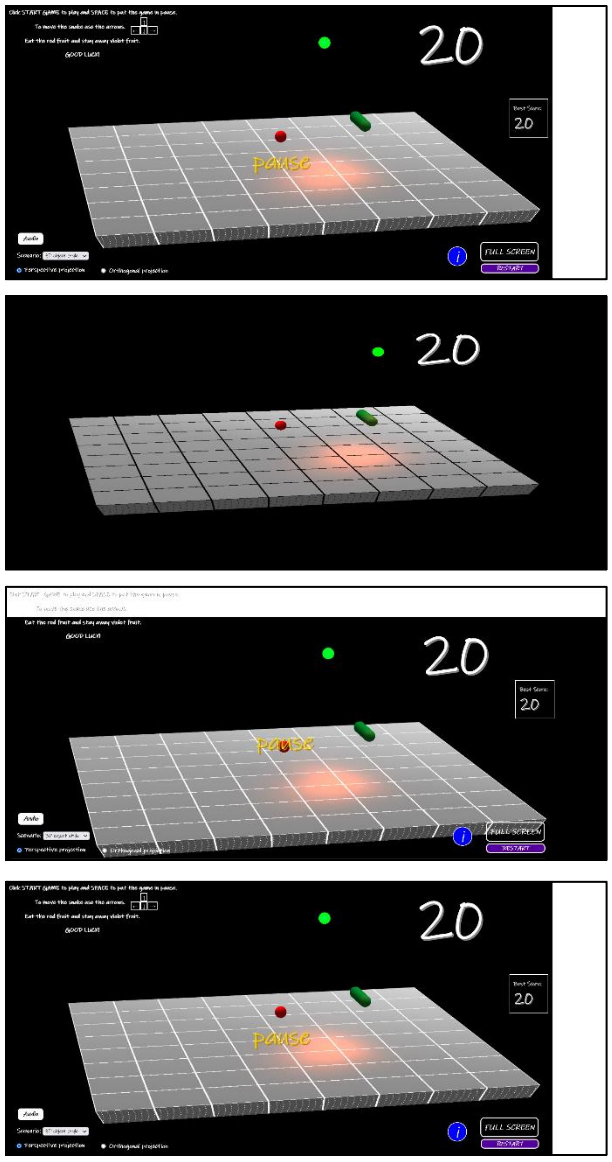


Figure 5 - In the first figure you can see the game screen before the full screen mode. In the second, in full screen mode. In the third, you can see the game screen after the first press of the ESC key, with the canvas slightly lowered compared to the initial position. Finally, in photo four, you can see the canvas in the original position after clicking the ESC key for the second time.

2.2.d INFO BUTTON

At any time, a player can click on the information button that opens the game manual and at the same time pauses the game.

2.3 AUDIO

In the development phase of the game it was decided to associate the various game phases with audio tracks that have been collected in a folder called Soundtracks. These tracks are in mp3 format and have been downloaded from YouTube and subsequently reworked to make them more functional. Each track is played or paused using the functions of the same name. The loop() function ensures that the music restarts once it is finished.

The audios used are: "Snake III JAVA game theme song.mp3", "Jeopardy Theme.mp3", "The Price is Right Losing Horn - Gaming Sound Effect (HD) .mp3", "Eat Munch 2 Sound Effect.mp3", “Pokemon ost-Lavandonia.mp3”,” Waterflame – Jumper.mp3”, “Bb Blues (110bpm) Backing track.mp3”. The first is the audio used as the sound base for the whole game. The second track starts when the game is paused. The third track is put into play when the game ends. The fourth track starts every time the snake eats food. The other tracks act as background sound for the different scenarios.

A button has been implemented that deals with modifying the volume value of the audio. This button shows the word AUDIO, when the sound is at its maximum volume, while once pressed it sets the volume of all the tracks to 0 and the word AUDIO is replaced by the word MUTE. A new click on the button resets the volume to maximum.

2.4 LIBRARIES

The libraries used for the realization of the project are collected in the "Common" folder in the github repository and contain basic functions that allow you to work with matrices, vectors and shaders. We thank Professor Emeritus Edward Angel for the material made available on the site https://www.interactivecomputergraphics.com/, taken from the book "Interactive Computer Graphics", from which the libraries come.

2.5 HIERARCHICAL MODEL

Hierarchical objects are important because they allow to progress beyond single objects like cubes or blocks to more complex objects that you can use for game characters, robots, and even modelling humans.

The hierarchical model created for our project includes a root element, i.e. the playing field, and four child elements, i.e. the snake, the sun, the fruit, which the snake can and must eat to grow, and the poisoned fruit, which, if eaten, results in the death of the snake and the end of the game. The creation and subsequent representation on the canvas are managed by the initNodes() function, which creates the hierarchical model, and by the traverse() function, which instead draws the individual elements and passes to the fragmentshader an integer value saved as "isSnake". This value is intended for the detection of the created object, in fact, depending on the value that this variable takes, the fragmentshader present in the Snake3D.html file is able to identify and differentiate its behavior for each node of the model. In practice, thanks to "isSnake" it is possible to color the playing field green while the fruit red, or apply a particular texture to one or more specific elements, simply by means of an IFELSE conditional construct. Thanks to the realization of the hierarchical model, it is possible to manage the objects created as a single model and this allows to rotate or translate in space, not only the playing field, on which the geometric transformation takes place directly, but also on all its child nodes. The keycodes useful for acting on the hierarchical model will be presented later.

2.5.a PLAYING FIELD

With regard to the root node, i.e. the playing field, it is important to note that it changes according to the scenario that each player selects. This involves not only changing the color of the pixels that compose it, but also the application or not of textures and other objects. It is necessary to focus on at least three scenarios that involve substantial changes to the playing field: the "grass field" scenario, the "no grass blades" scenario and finally the "3D object style" scenario. The first is the basic scenario in which the game opens and presents in addition to the lawn also a lot of blades of grass. These blades are obtained through the use of two specific functions invoked within the simulation()

function, which in turn is invoked within the init function. These functions are gen_blades() and myRandom(x, y). The first deals with creating and allocating information regarding the position and color of each blade of grass in the respective arrays; while the function myRandom(x, y) takes care of returning a float value between x and y. This was essential to be able to associate each blade of grass with a pseudorandom height and color in order to obtain blades of grass of different heights and shades of green, making the playing field more realistic. The implementation of a function called passOnBlade() manages the snake's passage over the blades of grass by reducing the height of the blades, just as if they were being squashed.

In the second scenario analyzed here, however, it is possible to see what is hidden under the blades of grass. In this scenario it is possible to better appreciate the textures applied on the playing field that highlight the edges and give it greater depth.

The third and last scenario presents a less realistic and more virtual situation, both in the colors and in the applied textures that highlight more the basic structure of the objects created. In this scenario, the contribution that light has on the scene and on the individual objects is highlighted.

2.5.b SNAKE

The snake is a subsystem of particles. At the beginning of the game it is made up of only 5 elements: one head element and four tail element. The movement of the snake and its growth are managed by the update() function, invoked within the render function. This function creates a further particle, temporarily called "head", characterized by having the y component of its position identical to that of the element 0 of the snake, but different x and z coordinates, depending on the direction of movement. At this point through the unshift(head) function the "head" particle becomes the new element 0 of the snake while the last element, that is the tail element, is popped. This sequence of operations causes the snake to move in the space; or rather, the snake does not really move, but the movement is reproduced simply by adding an element to the snake's head and eliminating one from the tail. Obviously if the snake eats a fruit the elimination of the tail element is not carried out and therefore the snake increases its length by one since it adds an element without eliminating another. One of the critical aspects of the work was precisely that of managing in the most correct way the moment of the collision between the snake's head and the randomly generated food. In fact, the threedimensionality of the objects and the pseudorandomness with which the food appears on the scene make it unlikely that at an instant of time t the head element of the snake and the element "food" share at the same time and exactly the same coordinates x and z. It was therefore decided to solve this problem by calculating the distance between the two particles under analysis with a margin of error below 0.0; that is, elements whose distance is below 0.0 are considered in the same position. To do this, such syntax was used

```
(snake[0].position[0]
food[0].position[0]).toFixed(1) == 0.0
```

```
(snake[0].position[2]
food[0].position[2]).toFixed(1) == 0.0
```

where toFixed(n) takes care of considering only n digits following the comma.

A similar method is also used to handle the case in which the snake's head element collides with another element causing the loss of the game. The management instead of the end of the game due to the overcoming of the boundaries of the playing field by the head of the snake takes place through simple inequalities and the association of a boolean value to some variables that if true involve the end of the game. This means that if the x or z component of the snake's head position exceeds the playing field limits, at least one of these variables becomes true, ending the game.

For the movement of the snake, we have opted for the convention "arrow keys", therefore a function has been implemented that changes the direction of the snake according to the key pressed. As in the classic snake game, it is not possible to pass from one movement to its extreme, but it is necessary to pass through an intermediate movement; that is, you cannot go to the right and immediately afterwards to the left if you do not first make a move up or down. At each change of direction one of the two movement components (x or z) are set to zero, while a nonzero value is assigned to the other, which tends to increase depending on the score, i.e. the level reached. A further difficulty takes over after level 250 where, in addition to the increase in speed, the first poisoned foods begin to appear which if eaten means the end of the game.

2.5.c SUN

During the animation it is possible to notice a particle detached from the playing field, but equally part of the hierarchical model, which represents the sun. This particle rotates around the playing field by following a sinusoidal trajectory in accordance with the displacement of light. In the same way as the snake, the movement for the sun is also managed through the update() function, which at each step is responsible for adding an element at the head of the sun object and eliminating the tail element. It was decided to add this object to the scene to better highlight the displacement of light in space and the consequences it entails on the playing field and on other elements. At the end of each game, the sun will set on the scene making it darker. This animation is managed once again by the update () function which, by keeping the coordinates of the sun node fixed along the X and Z axes, decreases the altitude of the sun at each step until a minimum value is reached.

2.5.d FOOD AND POISON

With regard to the objects "food" and "poison" we want to pay attention to the fact that at each level of the game they are generated, by calling the functions gen_food() and gen_poison(), and placed in random places, exactly as it happens in traditional game. In particular, the poison object, which now appears more and more frequently in the new versions of Snake, has the peculiarity of appearing on the screen only when a given threshold is exceeded and, upon reaching higher levels, even in greater quantities, always in order to complicate the game.

2.6 FUNCTIONS

This paragraph describes the main functions used for the realization of the project.

2.6.a INIT FUNCTION

The init() function in our JavaScript code is called when the script is loaded. Its purpose is to set the WebGL context and start rendering the content. It also takes care of implementing the functions of listening to events that affect the HTML elements, loading the shaders and sending the data related to the position, color, normal and texture coordinate buffers to the GPU. Also within it we find a first assignment of value to some variables, in particular to constants and the invocation to two functions: the first, configureTexture(), which takes care of configuring the three textures that will then be activated and used by the program and the second, simulation(), which is a link between init function and render function.

2.6.b SIMULATION FUNCTION

The simulation() function is used to create the various elements that will then be part of the hierarchical model. In fact, inside we find the calls to colorCube() and drawBlade() which are the functions that enter the first elements into the points, colors, textures and normals arrays. Note that calling the for loop on the drawBlade() function is essential for drawing and saving the position of each blade of grass. In the simulation() function, the snake elements, food, sun and poison are also created. Subsequently, the complete arrays are linked to their respective buffers in the form of subdata buffer. This is important, for example, to allow the growth of the snake and therefore the increase in the number of particles that make up it and whose coordinates need to be saved in order to be drawn on the scene. Finally, the call to initNodes() takes place, which takes care of forming the hierarchical model, and to the render function.

2.6.c RENDER FUNCTION

The render function takes care of the actual rendering of 3D objects on the canvas. It also allows you to continuously update the values and thus to animate the scene.

In our render function we find an important portion of code dedicated to setting the different scenarios following the choice made by the player. At the end we update and send the data regarding the model view matrix and the projection matrix.

The model view matrix was calculated using the lookAt(EYE, AT, UP) function, for which the vectors AT and UP, both threedimensional, are constant values, while the EYE value varies depending on the key pressed or thereafter dragging the scene using the mouse.

As for the projection matrix, we have chosen to leave the possibility to choose whether to use a perspective or orthogonal projection. The choice can be changed at any time by selecting one of the two radio buttons on the web page. Furthermore, each of the values that characterize one or the other projection can be modified thanks to a specific key.

Note that both matrices, the model view matrix and the projection matrix, are nonsingular matrices and are described by fourdimensional homogeneous coordinates: x, y, z, w.

Finally, the calls to the traverse(Id) and update() functions take place, which actually deal with drawing and animating the scene.

2.6.d UPDATE FUNCTION

In the update() function we find a first check on the state of the game, that is, if the game is in progress or has ended. Depending on the result of this check we have the possibility of seeing the word "GAME OVER" appear on the screen, followed by a further writing that changes according to the level reached, or the updating of the position of the snake and therefore the continuation of the game. The update on the position of the snake's head element and its possible growth have already been discussed in the previous paragraph, in the section dedicated to the snake, here instead we want to focus on updating the arrays of points and colors. In fact, to ensure the position update and therefore the animation of the snake it is necessary to completely clean up the arrays in question. Subsequently, the data relating to the elements that have not changed, i.e. the playing field and the blades of grass, which had previously been saved in alternative arrays, are passed to it, and then the other elements are allocated as had happened in the init function.

As for the case in which the first check the game is finished, the update function takes care of modifying the position of the light to associate a change of light at the end of the game.

2.6.e CHANGE_DIRECTION FUNCTION

The change_direction() function is primarily responsible for managing the movement of the snake following the pressure of one of the arrow keys. However in a more general form it deals with all the events that concern the pressing of a specific key. To implement this function, it was necessary to find the JavaScript keycodes list and associate a variable to each value. Pressing a key whose value has been linked to a variable involves the modification of one or more variables used in the rendering. For example, in addition to the aforementioned arrow keys, the numeric keys from 1 to 9 modify the values relating to the hierarchical model, allowing it to be rotated along the axes or translated in different directions; while the alphabetic keys modify the values relating to the view matrix of the model or the projection matrix, ie the characteristics of the camera. It is signaled that the ENTER key resets the values to the initial ones. For further details about the KEYEFFECT association, please refer to the relevant section

2.6.f HAS_GAME_ENDED FUNCTION

The has_game_ended() function has the task of returning a boolean value that reports whether the game is over or not. The checks that are made depend on the difficulty level of the game and the scenario envisaged. For example, until the score reaches a certain level and therefore the poisoned food does not appear on the scene, the check for the collision between the snake's head and any poison is not carried out. On the other hand, at any level, the control is carried out on the "bite of the tail", that is, if the head of the snake collides with any other part of its body.

Another check made is that the x and z components of the snake's head position must always be within the playing field. However, this control is no longer applied if the choice of scenario is "no limits". In this case, when the snake's head reaches one of the edges of the playing field, it will reappear along the same axis, but at the opposite end.

2.6.g TRAVERSE FUNCTION

The traverse(ID) function takes as input an identifying value of one of the nodes, at the beginning of the root node, and renders the function linked to that node, i.e. it calls the function that draws the hierarchical object on the scene and pass the "isSnake" value to the fragmentshader. Then the traverse(ID) function checks if the node in question has a child node or a sibling node and, if so, makes a recursive call with the node ID (child or sibling), otherwise it terminates.

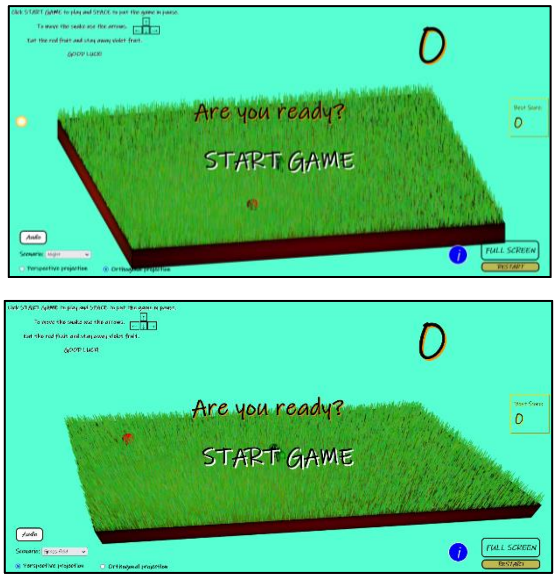


Figure 6 - Comparison between orthogonal projection, above, and perspective projection, below.

2.7 PROJECTIVE

The projection matrix is initially set on the perspective function (FOVY, ASPECT, NEAR, FAR) in almost all scenarios, except for the "2Dstyle " scenario, in which to reproduce a view similar to that of the traditional game projection matrix is set to the ortho function (LEFT, RIGHT, BOTTOM, TOP, NEAR, FAR).

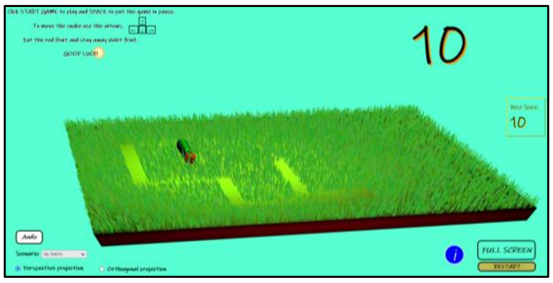
2.8 TEXTURES

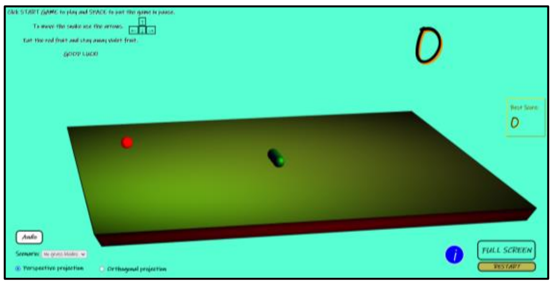
Three textures have been implemented. Texture0 takes care of creating the contours of the playing field. Texture1 is used to give depth to the color of the playing field. Texture2 draws a white grid on the playing field. The use of textures depends on the selected game scenario. The texture configuration takes place by calling the configureTexture() function which occurs within the init function and subsequently they are activated and associated with the active program.

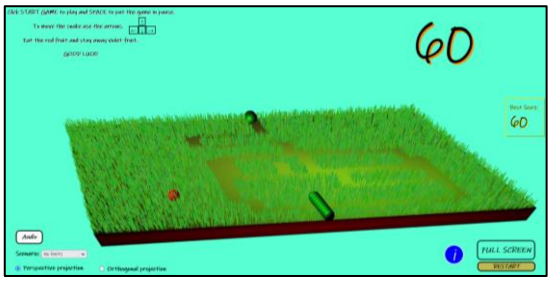
2.9 LIGHT

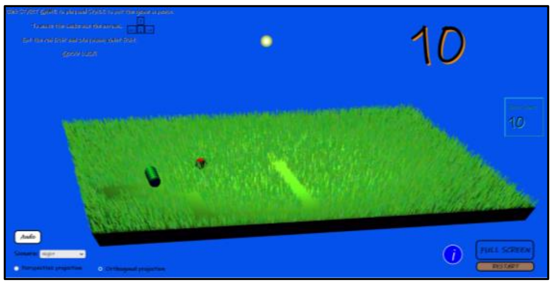
In this project it was decided to implement a color calculation technique based on the BillPhong model.

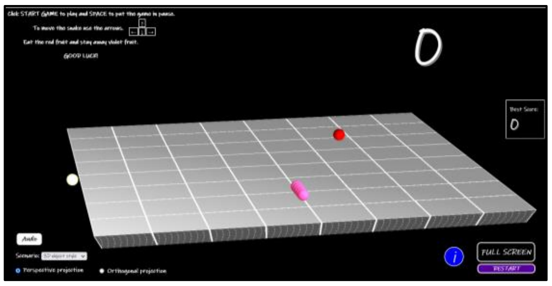
The implemented technique is a perfragment color calculation technique and is called Phong Shading. The basic idea is to determine the vertex normals and interpolate them across the edges and interpolate the edge normals across the polygon, then the modified Phong model is applied to each fragment. This means that, unlike Gouraud Shading, the illumination model is computed in the fragmentshader program, i.e. the vertex shader provides the normal and position data as output variables to the fragment shader. The fragment shader then interpolates these variables and calculates the color. The advantage of this approach is the greater realism. It is easy to see how the lighting effects in the case of a perfragment approach are more realistic, albeit with a higher computational cost.

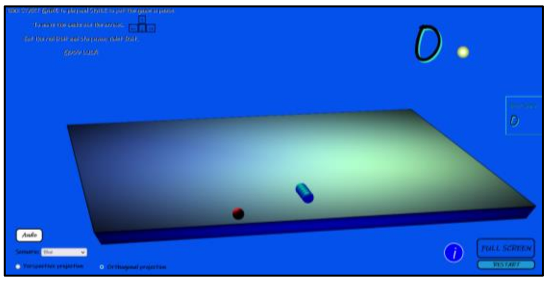












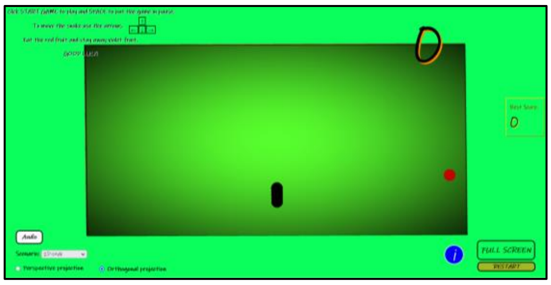


Figure 7 - The different scenarios. Starting from the top "Grass field", "No grass blades", "No limits", "Night", "3D object style", "Blue", "2D style".

2.10 SCENARIOS

The scenarios envisaged in this project are seven. Each of them is associated with an integer value "TextureCase" which, when passed to the fragment shader and to the render function, allows you to change the characteristics of the game.

The first scenario, the initial one, is "Grass field". In this scenario, all the rules of the classic Snake game apply. There are blades of grass on the playing field and the background color of the canvas is set to a light blue color.

The "No grass blades" scenario is identical to the previous one, but without the presence of blades of grass. To obtain it in the rendering function that deals with the playing field, you do not access the section dedicated to the design of the blades of grass.

The "No limits" scenario is based on the initial scenario in which, however, the control over crossing the boundaries of the playing field has been eliminated. In this case the game does not end and the snake reappears on the same axis, but on the opposite side.

The fourth scenario is "Night", on the playing field there are blades of grass and control over the borders is reestablished. The sun and light are made more silvery to recall the night effect and the background color of the canvas is set to dark blue.

The "3D object style" scenario completely changes the color of the playing field, on which not even blades of grass appear and applies texture2 to the ground. Furthermore, both the snake and the sun change their colors according to their position. The backgraund color of the canvas is black and all the colors of the HTML elements are changed to be more noticeable.

The "Blue" scenario is like the previous scenario, but all about shades of blue.

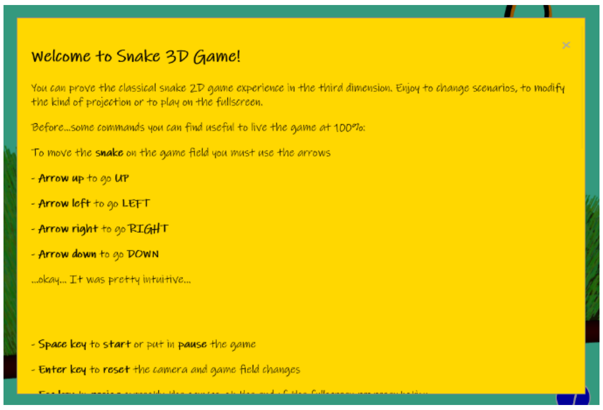
The seventh and final "2Dstyle" scenario is intended as a sort of homage to the classic game. The projection matrix is orthogonal, the playing field is arranged parallel to the camera, the snake is all black and the canvas is completely green.

In the Snake3D.html file it is possible to notice how the assignment of a specific value to the TextureCase variable corresponds to a different management of colors and textures, in order to obtain the game scenarios listed above.

For example, if TextureCase assumes a value of 5, ie the "Blue" scenario, it is possible to notice in the html file how, upon passing a specific vertex color, the fragment-shader reassigns another color. In particular, the vertex color vec4 (0.344,1.0,0.18,1.0), green, is reassigned a new color vec4 (0.517,0.884,0.940,1.0), blue.

2.11 TRACK CONTROLS

In the init function a portion of code has been implemented that takes care of keeping track of the dragging of the mouse on the HTML canvas element at each mouse click. In practice, every time this happens, the difference between the previous and final position of the mouse is calculated and then a vector of two elements called "currentAngle" is updated. Later in the render function, the view matrix of the model is rotated along the Y and Z axes according to the value assumed by "currentAngle"



Welcome to Snake 3D Game!

You can prove the classical snake 2D game experience in the third dimension. Enjoy to change scenarios, to modify the kind of projection or to play on the fullscreen.

Before...some commands you can find useful to live the game at 100%:

To move the **snake** on the game field you must use the arrows

- **Arrow up** to go **UP**
- **Arrow left** to go **LEFT**
- **Arrow right** to go **RIGHT**
- **Arrow down** to go **DOWN**

...okay... It was pretty intuitive...

- **Space key** to **start** or put in **pause** the game
- **Enter key** to **reset** the camera and game field changes

Figure 8 - Information box.

2.12 KEY LISTENERS

When the game loads, a button manual is displayed for players with specifics related to the controls. When this window is closed, the game can be started. The rules can be called up at any time via the information button at the bottom left. The following is the message that appears at the start of the game.

```
To move the snake on the game
field you must use:

Arrow up to go UP
Arrow left to go LEFT
Arrow right to go RIGHT
Arrow down to go DOWN


Space key to start or put in
pause the game

Enter key to reset the camera and
game field changes

Esc key to resize correctly the
canvas at the end of the
fullscreen representation

To rotate or translate the game
field on the canvas you can use
the following keys:

1 to translate game field towards
the x positive semiaxis
2 to translate game field towards
the x negative semiaxis
3 to translate game field towards
the y positive semiaxis
4 to translate game field towards
the y negative semiaxis
```

5 to translate game field towards the z positive semiaxis
6 to translate game field towards the z negative semiaxis
7 to rotate game field along x axis
8 to rotate game field along y axis
9 to rotate game field along z axis

To move the camera you can use the following keys:

A to increase x component of eye vector
D to decrease x component of eye vector
W to increase y component of eye vector
S to decrease y component of eye vector
E to increase z component of eye vector
R to decrease z component of eye vector
F to increase the depth of the camera
N to decrease the depth of the camera
+ to increase the field of view y-axis
- to decrease the field of view y-axis

2.13 END_TEXT

At the end of each game, based on the score obtained by each player, in addition to the word "GAME OVER", another choice appears written in a list of phrases defined as END_TEXT.

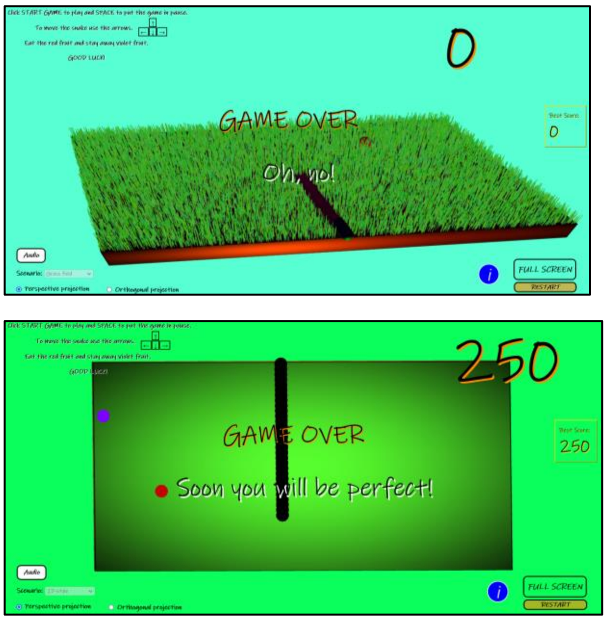

Click START GAME to play and SPACE to put the game in pause.
To move the snake use the arrows.
Eat the red fruit and stay away violet fruit.
GOOD LUCK!
GAME OVER
Oh, no!
GAME OVER
Soon you will be perfect!

Figure 9 - Some of the possible "GAME OVER" phrases present in END_TEXT.

3. USER MANUAL

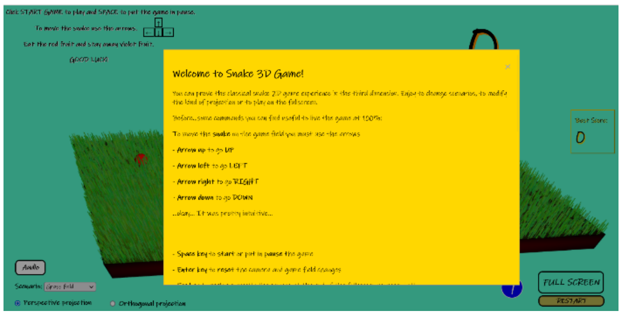
In this chapter we will deal with the operation of the player-side software. We will then describe how to actually play Snake3D. The game modes and action possibilities made available to each user will also be shown.

3.1 SNAKEGAME

Welcome to the 3D variant of the Snake game, one of the "Top 100 Games of All Time". The object of the game is to eat as much red fruit as possible but be careful not to bite yourself or crash. Use the arrow keys to move the snake in space in search of food and I recommend, if the food is purple, avoid it, it will not do much good for your snake.

3.2 SCORE

As in any self-respecting snake game, even in this one there is no goal to reach or a final level. The game continues as long as you can keep your snake alive. The fun is in improving yourself and trying to beat your own high score. At the top right you can see the current score and also your best score. Each time the snake stretches the score will be increased by 10 points. Try your best.


Click START GAME to play and SPACE to put the game in pause.
To move the snake use the arrows.
Eat the red fruit and stay away violet fruit.
GOOD LUCK

Welcome to Snake 3D Game!

You can prove the classical snake 2D game experience in the third dimension. Enjoy to change scenarios, to modify the kind of projection or to play on the fullscreen.

Before...some commands you can find useful to live the game at 100%:

To move the **snake** on the game field you must use the arrows

- **Arrow up** to go UP
- **Arrow left** to go LEFT
- **Arrow right** to go RIGHT
- **Arrow down** to go DOWN

...okay... It was pretty intuitive...

- **Space key** to **start** or **put in pause** the game
- **Enter key** to **reset** the camera and game field changes

Figure 10 - Initial game homepage with info box open.

3.3 INFO BOX

At the beginning of each game, an orange box will appear on the game's homepage, showing all the game controls that are useful for enjoying the game experience to the fullest. On the left of the box there is a bar to scroll the page. In order to allow a more complete adventure possible it is strongly recommended to read the text carefully. To close this window and start the game, you can click anywhere on the screen outside the orange page, or by clicking the cross at the top right. At this point you have reached the game homepage and can start playing.

Should you have any doubts during the game or would like to clarify yourself the game controls, simply click on the blue information button at the bottom left of the game screen. Once clicked, the orange box will reopen and you will be able to read all the information you need. Don't worry about the game, as soon as you will click the blue button, the game will be paused by itself, so you have plenty of time to read.

3.4 START GAME OR PAUSE

To start the game you can click on the word "START GAME" on the playing field, or by pressing the space bar. Always pressing the space bar it is possible, once the game is started, to pause it, in this way you will prevent your snake from continuing to move until it crashes or bites itself.

3.5 GAME CONTROLS

The game controls are illustrated in the information box accessible from the blue button located at the bottom left of the game screen. In general, the arrow keys are used to move the snake in space, the numeric keys to rotate or translate the playing field in space; while the alphabetic keys change the point of view with which you look at the playing field. Note that if you click and then drag the mouse on the screen, the playing field will move with you.

Below are the various game controls:

```
To move the snake on the game
field you must use:

Arrow up to go UP
Arrow left to go LEFT
Arrow right to go RIGHT
Arrow down to go DOWN
```

Space key to start or put in pause the game

Enter key to reset the camera and game field changes

Esc key to resize correctly the canvas at the end of the fullscreen representation

To rotate or translate the game field on the canvas you can use the following keys:

1 to translate game field towards the x positive semiaxis
2 to translate game field towards the x negative semiaxis
3 to translate game field towards the y positive semiaxis
4 to translate game field towards the y negative semiaxis
5 to translate game field towards the z positive semiaxis
6 to translate game field towards the z negative semiaxis
7 to rotate game field along x axis
8 to rotate game field along y axis
9 to rotate game field along z axis

To move the camera you can use the following keys:

A to increase x component of eye vector
D to decrease x component of eye vector
W to increase y component of eye vector
S to decrease y component of eye vector
E to increase z component of eye vector
R to decrease z component of eye vector
F to increase the depth of the camera
N to decrease the depth of the camera
+ to increase the field of view y-axis
- to decrease the field of view y-axis

3.6 SCENARIOS

At the bottom left of the game screen there is a drop-down menu that allows you to change game scenarios to guarantee you more and more personalized fun. There are 7 options and I invite you to discover and try them all because each has its own particularities.

The starting option is "Grass field", but if you prefer to play without the blades of grass to bother you, I suggest you select the second scenario "No grass blades".

The third "No limits" scenario allows you to ignore the edges of the playing field and can be a good way to increase your personal score. Whenever your snake finds itself with its head on one of the edges, it will reappear on the opposite side of the playing field as if nothing had happened. But beware that if you bite yourself or eat the purple fruit the game will end anyway.

The "Night" scenario is for you if you prefer less bright colors and a slightly darker atmosphere.

"3D object style" is a scenario different from all the others. You will find yourself in a more essential reality that recalls some virtual-futuristic worlds of the games of yesteryear. Pay attention to the color of your snake, depending on the area in which it will be found you will be able to see a change.

The fifth scenario "Blue" is simply a game mode that is based on shades of color: the whole screen will be colored in blue.

The last scenario "2D style" is the scenario that best suits the nostalgic of the game. It will allow you to relive the emotions of the past.

3.7 RESTART BUTTON

At the end of each game, if you are not satisfied with your score and think you can do better, you can always click the restart button. This way the adventure will restart and you can improve your score or just have fun with other scenarios.

However, note that it will not be possible to restart the game until it is over.

3.8 AUDIO BUTTON

On the left side of the screen, in addition to the possibility of changing the scenario, it is possible to mute the background music or restart it by pressing the AUDIO button.

3.9 PROJECTIVE RADIO BUTTONS

Have you ever experienced the difference between perspective projection and orthogonal projection? Now you can. In fact, by clicking on the appropriate radio button at the bottom left you can compare the two projections and choose the one that suits you best.

3.10 FULL SCREEN BUTTON

If you are tired of playing in small screen mode, you can, by clicking on the FULL SCREEN button, switch to full screen mode. However, when you want to exit this mode, remember to click the ESC key once to return to the main screen and click it again to return the game screen to its original position. In case of doubt, this advice is also reported in the information box accessible via the appropriate button.

3.11 CONCLUSION

That's all. Have fun challenging yourself and your reaction skills and try to improve yourself more and more. Good luck snake-player!