Interactive Graphics - Final Project
Snake 3D

Leonardo Ospizio - 1808991
Francesco Riccardo Tassone - 1814263

July 25, 2021

1 Introduction

Snake3D is the reinterpretation of the famous game created in the 70s, very popular until the 90s. The original game predicts that the snake can move around the floor by eating the food generated in random positions so that the length of the snake grows as the game progresses. The game ends when the snake hits one of the side wards or itself.

2 Snake3D

In our implementation, the snake moves on the faces of a cube and in addition to eating food it can collect objects that activate extra features that make the game experience much more interactive and hectic. As there are no side barriers, obstacles have been added that, if hit, end the game. All the objects with which the snake can interact have different characteristics, they can move and sometimes be destroyed. We have implemented several bonuses. The score bonus increases the multiplier to the score obtained when eating an object in the environment. The lucky bonus can spawn other items, obstacles, food or other bonuses. Finally, the invincibility bonus allows the snake to destroy obstacles in the environment.

2.1 Game modes

We have foreseen different game modes:

- Custom, gives the user full freedom to choose such as the size of the environment, its appearance, the elements to be created, the level of difficulty and so on.
- Regular, there are several worlds to tackle, the player wins the game when they pass all the levels in the selected world. In this case the environment is not customizable.

2.2 User manual

By accessing the snake3D page, a loader loads all the required resources to avoid annoying delays in the following moments. Once the page is loaded, a control panel allows the player to choose his username and configure his game, if he decides to play in custom mode, or choose the world to face, if he decides to play in regular mode. As soon as the configuration is saved, the user can start playing by clicking play. The player can move the snake using the w/a/s/d keys. When the game is over, a report message about the game just played is shown. clicking continue takes you back to the main menu and a new game can start. We suggest to play at least 2 games, one with the Standard texture pack and the other with the Pack 1.

3 Technical manual

3.1 Technologies

The game is implemented using Three.js (Rev 129) for the graphic part, Tween.js (Ver 18.6.4) to generate the interpolated data for the animations and the MV.js provided library for the linear algebra. The user interface has been implemented using Bootstrap v5.0

3.2 Game environment

The game environment manages the interaction between the snake and the other object on it. It keeps track of the position of Snake, Obstacle, Bonus and Food entities.

3.2.1 Model

The building block of the environment is an invisible cube that acts as a container for the visible cube in which the snake the other entities are placed. Each entity is contained in a generic 3d-object used to correctly placed the entity in the environment container. This object contains the entities mesh and sometimes also a light object.

3.2.2 Data structure

As mentioned earlier, the entities can change their environment position, can be spawned, modified, moved, and destroyed during the game. In order to maintain consistency we have implemented several data structures that can be modified by low-level methods. The environment is composed of a $width \times height \times depth$ matrix of CubeCell entities. Each cell wraps an entity (Snake, Obstacle, Food, Bonus). The environment matrix contains all the alive entities. The environment manager uses a coordinate manager that keeps track of available and unavailable cells, to access the environment matrix and execute operations. The main operations executed on the environment are:

- spawn entity
- modify entity
- move entity
- destroy entity

These operations put the entities in the respective queues in order to update if needed the views, execute animations, and so on.

3.2.3 Animations

The animations implemented for the environment and respective entities are the following:

- Entity movement: a random direction and offset is generated by the environment manager. If the movement is applicable the entity is moved from the current location to the generated one.
- Entity destruction: the environment manager removes the entity mesh from the scene and spawn randomly small particles that move from the destroyed object position in random directions, these geometries becomes gradually invisible and after a random amount of time are removed from the scene.
- Environment generation: whenever a new environment is generated, the environment appear small and became gradually larger rotating on all its axes.

3.3 Snake

3.3.1 Model

The snake is a dynamic hierarchical where the head of the snake is added to the environment mesh as a child and each node is created dynamically and it is added behind the last spawned one. Each node, including the snake head, is composed of a container, node, and light. The container is an empty object without geometry and material, it is used to decouple the relative position of each node (w.r.t to the previous container) and the orientation. In this way changing the position of the container i affects all the containers between $i + 1$ and N. Any rotation applied to a node, instead, does not affect the rest of the snake.
The node is the visible mesh that contains the geometry and material selected. It is added as a child to the corresponding container. Each node shares the geometry and material in order to reduce the amount of data (vertexes, materials, and textures) passed to the GPU and reduce bottlenecks when the node is created.
Lastly, we have a point light added as a child to the node. Its proprieties depend on the material applied to the node. [3.6]

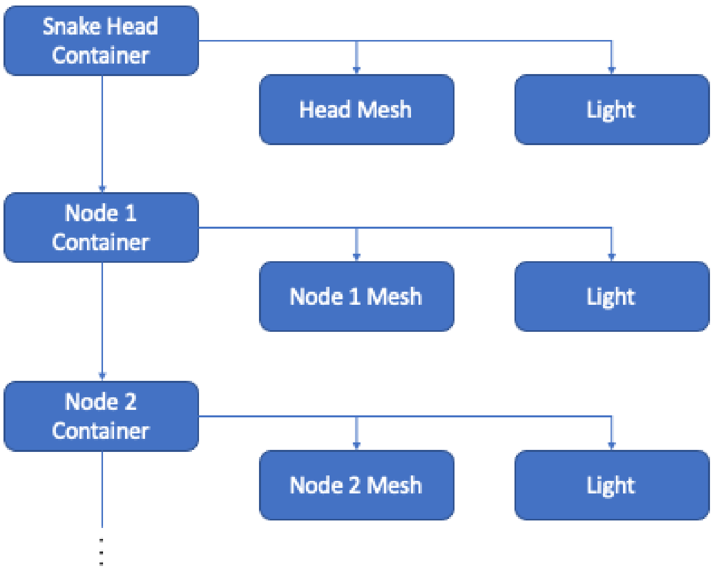


Figure 1: Snake hierarchical model

3.3.2 Data structure

The snake is handled by three fundamentals elements: Controller, Snake entity, and the AnimationHandler.

Controller
The Controller is responsible for handling the keyboard events, scheduling the snake movement, and managing the world orientation.
The world orientation is stored as two vectors representing the "up" and "right" axis w.r.t the viewer. When the user presses w/a/s/d the correspondent direction function is triggered, and the new snake direction will be scheduled along the chosen axis (e.g: if the user wants to go to the left the new snake direction will be along the negative "right" axis). After choosing the direction, the Controller computes the new snake position, check if it is feasible and finally sends it to the Snake entity and updates the position of each node in the environment. If the new position is out of the environment borders, i.e the snake is at the end of one cube face, the position is discarded and the Controller proceeds to compute the new world orientation. The new coordinates system CS_n is computed as follow:

$$CS_n = R(r, \theta)CS_o$$

Where:
CS_o is the old coordinate system.
$R(r, \theta)$ is the rotation matrix of an angle θ around an axis r.
r and θ are chosen accordingly to the snake direction and the world axis. In particular, r is the "other" axis respect to the snake direction axis and θ depends on the snake direction sign. (E.g: if the snake is moving along the positive world "up" axis the rotation will be of 90° around the world "right" axis).
Finally, the new position is computed using the new coordinate system.

Snake Entity
The core of snake structure is the Snake entity. It contains the list of "SnakeNode" (including the head) and it is responsible for scheduling the snake movements and animations. Each "SnakeNode" contains the position, direction, and the reference of the node mesh. When the Snake entity receives a new position from the controller it schedules the head position translation and, if needed, the relative position change of each node.

AnimationHandler
All the movements (translations and rotations) are managed using an event queue. Each element of the event queue is a list of Synchronous events, i.e when the event at time t is triggered all the animations inside the first list in the event queue are activated. Each Synchronous event contains at least one head position change, the relative position of the nodes changes and all the animations to be executed at the same time. These movements generation will be treated in the next section. [3.3.3]

3.3.3 Animations

The snake movements, internals and externals, are generated when the Controller sends to the snake a new position, but they are not executed yet. The execution of all the Synchronous

events at time $t + T$ is done when the previous animations at time t finish. Thanks to this system the user can send an input at any time between t and $t + T$ and the snake will execute it only when it arrives at the end of the current environment cell.
There are three main groups of animations: head translation, nodes relative translation/rotation, and node creation.

Head Translation

The head translation is the animation that changes the head position, and therefore all the snake nodes positions, in the environment. It is generated using Tween.js to interpolate the current position and the target position in T seconds. Then the animation is added to the events to be activated at time $t + T$ by adding it to the head of the event queue.

Nodes Relative Translation and Rotation

If the head new translation direction is different from the current one, for each node a relative translation animation is created. Indeed, each node should continue its movement until it reaches the cell where the head had changed its direction. To do it a new translation animation is computed as two simultaneous movements. The first one has the purpose of keep moving the node where the animation is applied (and consequently all the following ones) in the same direction as before. The second one is to "compensate" the movement of the previous nodes in the new direction.
To the relative translation animation is associated also a node rotation. The rotation is computed like the world coordinates change using the axis-angle rotation matrix:

$$R_n = R(r, \theta)R_o$$

Where:
R_n is the new coordinate system.
R_o is the old coordinate system.
$R(r, \theta)$ is the rotation matrix of an angle θ around an axis r.
r and θ are chosen accordingly to the node current direction and the new one. $r = d_o \times d_n$ is the cross product of the old direction vector and the new one and θ is the angle between them.
After the generation of the animations, one for each node, the event queue is updated. The animation of the ith node is added in the ith position of the queue. [Fig.2] In this way, the animation will be executed at time $t + iT$. The result is that the node changes direction only when it arrives at the environment cell where the rotation was commanded by the user.

Node Creation

When the snake eats a Food a new invisible node and few animations are generated. In particular for each node already present in the scene a small animation that changes the node scale is added to the queue using the same system as before where the ith animation is added at time $t + T$. After that, the creation animation is added at time $t + NT$ where N is the snake length. It is composed of a small delay to start the animation at the center of the environment cell and it scales the new node from scale 0 to the final value.
Because the new node cannot know all the movements scheduled before its creation when its animation starts the position is calculated as the translation from the origin of the previous node along the negative direction axis of it.

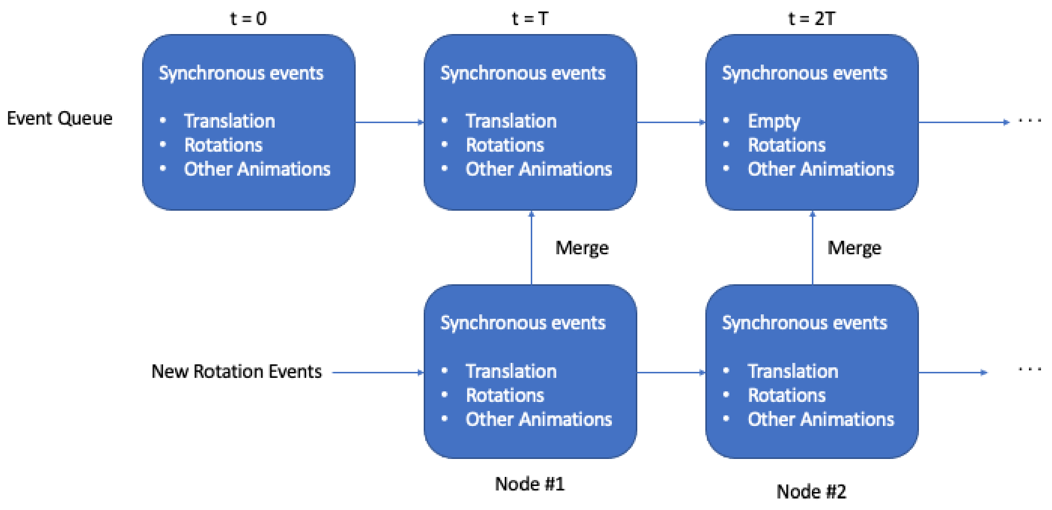


Figure 2: Adding a rotation to the event queue

3.4 Camera

The camera is a simple perspective camera child of a container attached to the scene. The camera container has as children also a point light and the score counter. In this way when the container moves the camera, the light and score counter moves accordingly. This choice was taken to decouple the position and orientation of the elements, as before. The camera container position is computed starting from the "up" and "right" vectors and computing the cross product. The result is the axis along the camera should be placed. Then a slight offset is added to the result along the "up" and "right" axes to have a better view. The movement is computed using Tween.js for the interpolation.

3.5 Engine

This module is the core of the system. The game engine takes care of managing the interaction between the user and the game. It contains the main modules such as the MatchManager which manages a single match or a series, the ScoreManager which keeps track of the score and the EnvironementManager shown above. The initialization of the various modules, loading of resources, reading and saving of configurations and management of the handlers of events triggered by the user are activated during its creation.

3.6 Graphics and textures

There are two texture packs: "Standard" and "Pack 1". The former is completely build by hand specifying each geometry and material of each element in the game. The latter is inspired by a very popular game with some pre-built models and some textures.

3.6.1 Texture Loader

The textures are loaded right after the page load, in this way every texture is loaded before the beginning of the game and it is possible to change them between any round. At the beginning of each round, indeed, the Game Engine chooses the right geometries and materials to be applied to the elements. Each element of the same category (e.g. Obstacle, Food, SnakeNode, etc...) shares the same geometry and material. This approach has two main advantages: the first one is that the game needs less resources because all the vertices and the material (color, texture, maps, etc...) are loaded in the GPU only once, the second advantage is that it is possible to change any parameter at any time once for all the elements that have the same structure.

3.6.2 Standard Pack

The Standard pack is built around two principles: simplicity and light effects. The environment is semi-transparent and slightly lucid to allow the user to see all the elements on the cube. The snake is made of solid cubes with an emissive component and a point light inside of the same color as the cube. The obstacles are similar to the environment cube. The food is a simple red sphere made of a very shiny material, in this way is possible to see the camera and snake lights reflections. Each bonus is a very luminous golden element with a shape that represents the bonus type.

3.6.3 Pack 1

This pack, inspired by a famous game, shows the capabilities of the system. Indeed, it is possible to add to the game any texture pack simply by adding a configuration, the resources, and some fine-tuning. The environment is a lawn made by a color map that defines how the color should be applied and a normal map for the lighting. Both the color and the normal maps are external resources. The obstacles are cubes with an image texture applied to them. (The image is an external resource). The Food and the bonuses are externals models.

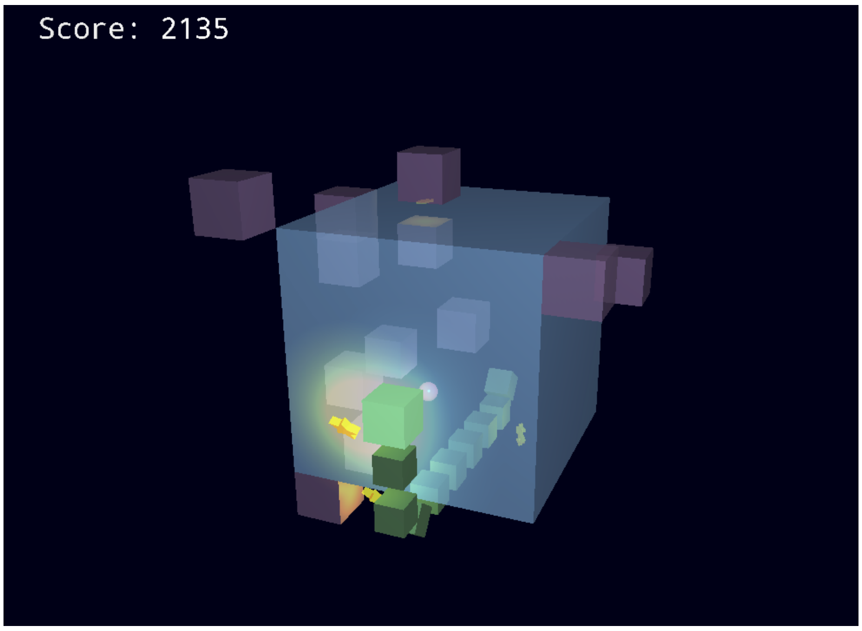


Figure 3: Standard Pack Environment

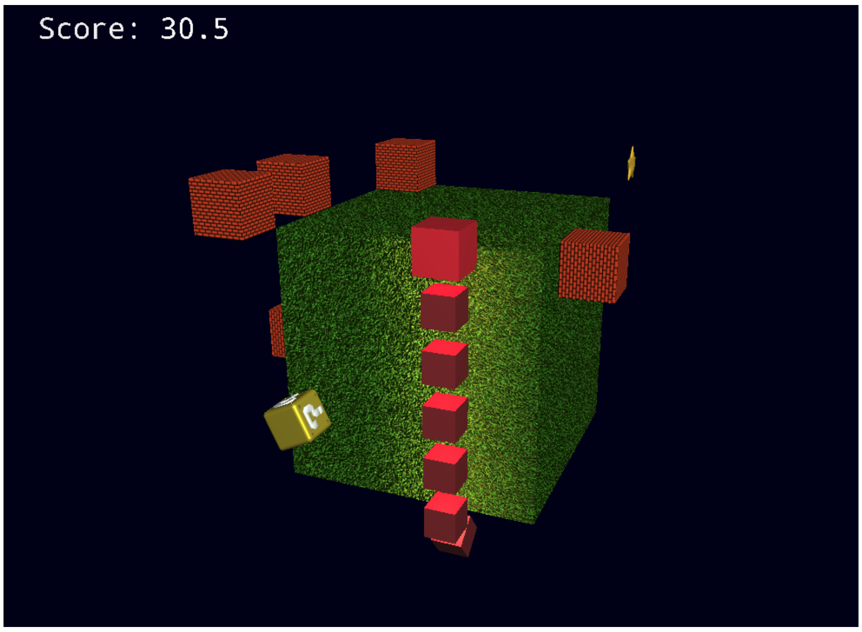
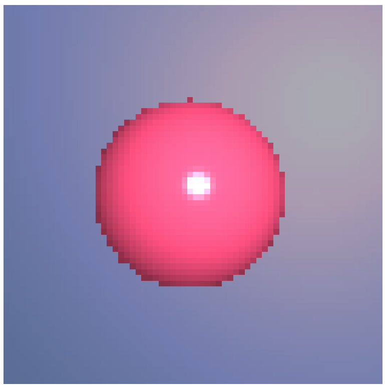


Figure 4: Pack 1 Environment


(a) Standard

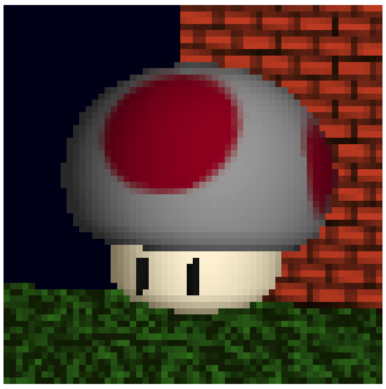
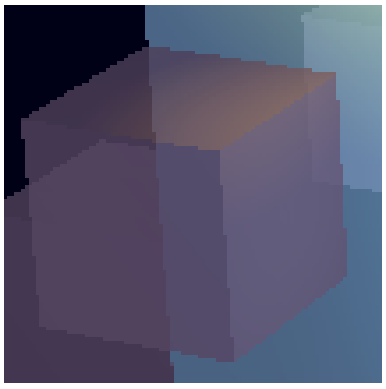

(b) Pack 1

Figure 5: Food


(a) Standard

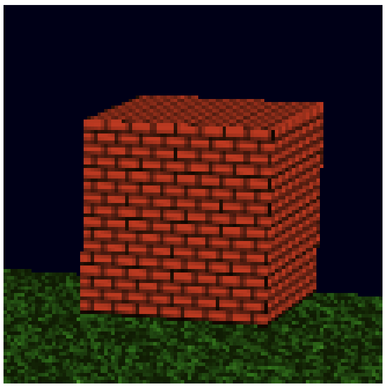
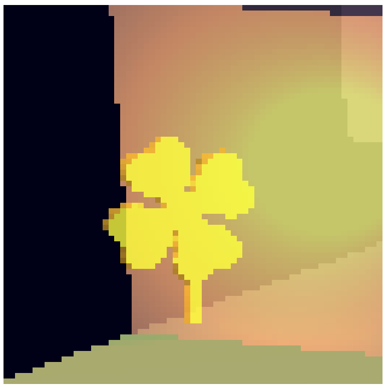

(b) Pack 1

Figure 6: Obstacle


(a) Standard

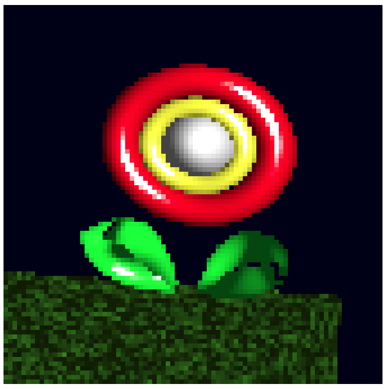
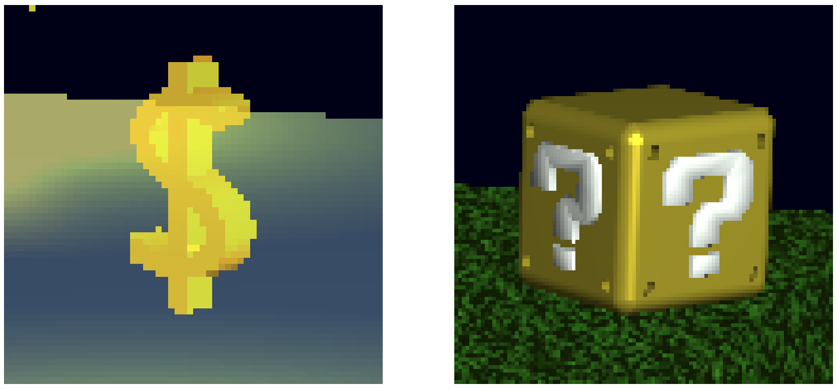
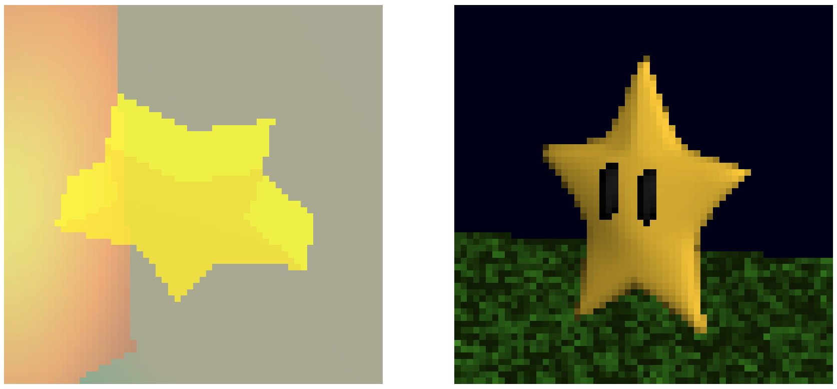

(b) Pack 1

Figure 7: Lucky bonus



(a) Standard (b) Pack 1

Figure 8: Score bonus



(a) Standard (b) Pack 1

Figure 9: Invincibility bonus