Chocobo Farm

Team
Francesca Fiani

Course
Interactive Graphics

A.Y.2019-2020

SAPIENZA
UNIVERSITÀ DI ROMA

Index of contents

1. Introduction and tools

Chocobo Farm is an interactive 3D browser game developed in the context of the Interactive Graphics course. In it, the player impersonates a farmer who has to compete with other farmers over who produces the best-selling crops to feed the titular characters, the chocobos, animals resembling big yellow birds. The setting is that of two neighboring farms, each with its 6x8 field. To win, the player must avoid the 5 mines placed in his field by his opponent (the pc or another human player, depending on the chosen game mode). You can win either by planting all the 35 crops avoiding the 5 mines, or more feasibly by having your opponent dig up a mine before you. Hitting a mine means your stable's door will blow up, your chocobo will flee and steal all the crops you planted up to that moment. To access it, it's sufficient to click on the following link:
https://sapienzainteractivegraphicscourse.github.io/final-project-francesca-fiani/home.html

The main environments used are HTML for the pages' layout and JavaScript for the game mechanics, with a small use of CSS for the style of the popups. In particular, the JS libraries chosen for the project are Three.js for the basic handling of the graphic elements (models, lighting and shadows) and Tween.js for the animations of the models and the camera movements.

Moreover, a heavy use of Blender allowed the models to work as skinned meshes through rigging and reprocessing of the models. In particular, the models Chocobo2.0 and Quina presented a confusing mesh structure, made of unconnected and often overlapping sub-meshes. They both were reprocessed in the following way: elimination of duplicates, separation by loose meshes and finally merging by distance with a very small distance. The processed Chocobo2.0 model resulted this way in a single mesh, while the Quina model still

presents some unconnected sub-meshes that doesn't affect the rigging. Then a manual construction of a skeleton in Blender allowed those two models to work as hierarchical models once loaded in JavaScript. The .blend files are available in the GitHub folder of the project:
https://github.com/SapienzaInteractiveGraphicsCourse/final-project-francesca-fiani
All the sources for the used models, images and sounds can be found in the last chapter.

2. Home page

Upon clicking on the given link, the player reaches the home.html page. The player is greeted by warning about the game's background music, which is by default turned off. After clicking the only button in the popup, a chocobo noise is played automatically. This allows the application to load at will any sound, bypassing the security issue in many explorers which disallows automatic reproduction of music. The structure for the popup is defined in the CSS 1vs1.css, but the use of 1vsPC.css would have been analogue (the CSS coincide in what regards home.html). It must be noted that nearly all the dimensions of the application (font sizes, margins and images' dimensions) have been expressed in viewport height and width, in order to make the application as much as possible correctly visible in any window ratio. The perfect ratio, in which the game has been developed, is 1920 x 1080. By clicking on the play now button, a new window pops up, asking the player to choose the game modality. If the player chooses single player, the window is redirected to 1vsPC/1vsPC.html and here the player has to play against a simple computer, whereas if the player chooses 1 vs 1, the window is redirected to 1vs1/1vs1.html and the player will compete against another human player (as a side note, this version was the one heavily used for testing the game implementation). A cancel button is present in this window, if the player wants to close the window, and clicking again on play now will prompt the window again. The page, like all the others, is linked to a JavaScript file called utilities.js, in the js folder, which contains all the functions useful to the handling of the html (like the appearance and disappearance of popups, their contents and the background music's on/off).

3. 1 vs PC mode

In this page, the true game is implemented. Besides the usual CSS, the page is linked to a module type JavaScript called 1vsPC.js, which all the modeling and game mechanics. When the page is opened, the player is greeted by a loading screen which will be unremovable until all the 3D elements of the page have been loaded. Since the chosen models are often heavy, the loading time can be quite long (up to a couple of minutes). Other than the setting of the game, which will always be visible and consists of 121 scene elements (which include camera, lights, helpers, planes and models), the game loads three additional models (namely Quina, turnip and rocks) which are deleted from the scene once the loading is complete in order to avoid lags during the game due to loading. In fact, once a model is retrieved from the server it's saved in the cache, and therefore every time an instance of the same model is requested, the game retrieves it from the cache instead of the server, making computation faster. After the loading of the setting is complete, a 4-page guide for the game is loaded in the usual popup structure, complete with screenshots of the game to exemplify the functionalities. By either clicking on the cancel on the top-right of the popup or reading all the four pages (which can be traversed freely) and clicking on the let's play button, another popup signals the start of the mine positioning phase of the game. After closing the popup, the setting is finally visible, so let's stop to analyze it.

The scene has a fog factor implemented, which makes objects far away from the camera (positioned on a heightened plane and pointing to the center of the scene) look greyer the more the distance is. Then, a hemisphere light simulates the sky and ground colors, and an orange directional light is positioned high on the y axis and shifted both on the x and the z axis to simulate sunlight. A camera helper and a directional light helper have been implemented for testing purposes but

are currently not visible. About the models, each of them has been loaded through a gltfLoader, and its structure has been traversed in order to find the mesh elements and enabling both shadow casting and shadow receiving on them. In the case of the Chocobo models, after the loading the animation which will be used in the later phase of the game has been implemented through tweening. First, the bones have been extracted from the armature through the custom function traverseTree, which creates a dictionary associating the name of the bone with the bone itself. Then, to create the animation the tweens for the rotation of the bones and the position of the model have been chained and linked to a specific tweening group (groupChocobo1 and groupChocobo2). They will be activated only after one of the players have lost, as specified in the render() function.

There are also two buttons to the right of the screen. The one on the top is a redirect to the home page, while the one on the bottom handles the audio. By default, the audio is off, but by clicking the button the audio is activated and the icon changes. A second click deactivated the audio and returns the icon to the previous one.

Back to the game, after closing the popup the player must place the mines in the PC's field. By now the mines have already been placed by the PC through a function, initializeMines, which chooses five random positions in the mines' matrix of the player (of course if one of the chosen values already contains a mine it recomputes the position). In the top-left of the page a custom version of the popup window is tasked with keeping count of how many mines have been placed by the player by consulting the nminesp2 value (initialized to 0). Note that while in the game phase a p1 in the variables corresponds to the player and a p2 to the PC, in the mining phase a p1 corresponds to the PC and a p2 to the player (this is because the consistency is with the field and not with the player, since for

example the player sets the mines in the second field, the PC's). The fields have been initialized in 40 different patches separated by a grid instead of a single patch in order to allow picking. For this purpose, two new scenes have been created, called pickingScene1 and pickingScene2, and the fields have been added to them patch by patch, with each patch transparent in this new scene. Moreover, each patch has been associated with an incremental id in a specific array called idToPatch1/idToPatch2. Then, a GPUPickHelper class has been created. In it, four methods handle the picking of the patches. For now, we'll see the two methods regarding the mining.

The first method is pickmines, which handles the hovering. The method takes the single pixel pointed by the camera, finds the corresponding pointed patch (if a patch is pointed) and if no mine has been placed in the patch it colors it in yellow in the pickingScene. This method is constantly called by the render() function until the mining phase ends. The second method is clickmines, which handles the clicking. The method first works like the pickmines, but instead of coloring the patch, if the mines is not set in that patch it places a mine, both analytically by placing a 1 in the minesp2 matrix and physically by placing a 3D model of a pile of rocks, and changes the counter. Moreover, if all the mines have been placed, all the models of the rocks are removed from the scene, the camera is moved to frame the other field (through the function moveCamera, which applies tweening to the camera position and lookAt) and another popup signals the start of the game phase. This method is called only when the window, through an event listener, reads a mousedown event.

After the mining phase is over, the game phase starts. The top-left counter becomes a track for the turn, and the player has to choose a patch where to plant a gysahl green, symbolized by a turnip. The hover and picking works similarly to the mining phase, but with some small differences. For this reason, two more

methods have been added to the class GPUPickHepler, pick and click. The only difference between pick and pickmines is that pick checks if a patch has been clicked (checks if the corresponding value in the matrix clickedp1 is 0 or 1) to decide if it must be painted yellow instead of checking if a mine has been placed in the patch. The click method, instead, is completely different from clickmines: after checking if the patch has been clicked, it checks if a mine has been dug in the patch. If there's no mine, it changes the corresponding value in clicked to 1, calls a function plantGysahl (which after disallowing the hovering of the patches handles the animation of Quina, places a turnip model in the center of the patch and calls moveCamera) and changes the value of turn. The moveCamera function will also call the function choosepatch, which handles the enemy turn. In particular, it keeps choosing a random patch until it finds one that has not been clicked, and then depending on the value of minesp2 reacts like click. If, instead, there is a mine, it calls the function explode after signaling the loss of the player by setting the exploded variable to 1 (if the loser is the PC, the value is set to 2). An additional note on the function placeGysahl: if 43 turnips have been placed, and consequently all the mines have been avoided, the player automatically wins, and a window asks him if he wants to play again or return to the home screen. Since the first player is always one step ahead of the second, only the first player can win by placing all the turnips.

The function explode first deactivates the home and audio button, sets the animating flag to 1 in order to stop the patch hovering, then performs a cut version of the planting animation. Once the animation is complete, it stops the music, creates a white plane in front of the camera, tweens its opacity and plays an explosion sound to simulate the explosion of the mine. Finally, the game over music is played, the chocobo is animated to run out of the stable until the border of the screen and the turnips are picked up from the ground, disappearing after

some milliseconds. Once the animation is complete a popup prompts the player to choose between playing again, reloading the page, or going back to the home screen.

Since some aspects of the render function have not been analyzed, we'll see them now. The render function handles all the tweens and picking by various checks on the game variables, like animating, cameramoving, exploded and turn, and chooses accordingly what tweening or method must be executed. Moreover, the render handles the setting and clearing of the pickPosition anytime the mouse is moved (the pickPosition will then be used to determine which patch has been clicked and/or hovered).

Moreover, the 1vs1.js module contains an unused function, dumpObject, which was used during testing to understand the structure of the models loaded.

4. 1 vs 1 mode

Since this page is very similar to the 1vsPC one, only the main differences will be analyzed. Please note that for an optimal experience the two human players, playing on the same pc, should not cheat and look at each other's mining phase. The loading phase is identical, and while the text and images in the guide differ the structure of the popup is unchanged. Once we get to the mining phase, however, there are some small changes. Instead of having the current P2 as a PC opponent, since it's a human player the manual mining, which in the 1vsPC version was possible only for one player, has been duplicated. For this reason, in the GPUPickHelper class the old methods pickmines and clickmines handle the mining in P2's field as pickmines2 and clickmines2, while a second version pickmines1 and clickmines1 has been added to handle the mining in P1's field. Their functioning is identical, and the only differences between the two versions are the coordinates which will be used for the model placing and the variables influenced. Moreover, the function initializeMines has been removed and is not called anymore before calling the main() function.

The same thing can be said about the game phase, in which the automatic play of P2 as PC has been substituted with a manual play. For this reason, another two methods have been added to the GPUPickHelper class, while the function choosepatch has been removed, and is not called anymore by moveCamera. The old pick and click became pick1 and click2, and pick2 and click2 have been added to mirror them in P2's field with the same functioning but changed coordinates.

The addition of more methods to the GPUPickHelper class means more checks in the render() function. In particular, two more checks have been added for the picking in the pre-existing ones by means of checking the turn value (which equals 1 for mining in P1's field, 2 for mining in P2's field, 3 for P1's turn and 4

for P2's turn) and two for clicking in the click() function also by checking the turn value.

The rest of the code is identical, and for this reason will not be further analyzed.

5. Sources

Models (all models come from SketchFab):

- Bush: https://sketchfab.com/3d-models/photorealistic-bush-8db54bf299954daa9ee29b233e923672
- Chocobo: https://sketchfab.com/3d-models/ff-vii-chocobo-for-3d-print-897c70e0a023484d8cd2d87bd0f83c06
- Fence: https://sketchfab.com/3d-models/handpainted-fence-free-agustin-honnun-9c2fb0b1afaa450f97ac9f6217ed6dc9
- House: https://sketchfab.com/3d-models/house-1-individual-export-f9f622f4a80c40ac9ddfec572ad4142e
- Quina: https://sketchfab.com/3d-models/quina-final-fantasy-ix-fan-art-a4f65f440bf5423181f341ae1433ab2b
- Rocks: https://sketchfab.com/3d-models/rocks-5a0ecd41663c4072abb224806151f7cb
- Tree: https://sketchfab.com/3d-models/old-tree-5793911821d04f73926ac4ab1d46727a
- Turnip: https://sketchfab.com/3d-models/turnip-4a387cc30acc49e59e243cf192ec2870
- Well: https://sketchfab.com/3d-models/old-well-7fa29793eff14a7587ee0af15446a675

Images:

- Clouds background: https://eskipaper.com/cloud-background-1.html
- Grass texture: https://www.sketchuptextureclub.com/textures/nature-elements/vegetation/green-grass/green-grass-texture-seamless-19524
- Ground texture: https://sketchfab.com/3d-models/ground-square-20x20m-1ab2168885d34640aefb5b37e1e747aa
- Cancel button: https://icons8.com/iconizer/en/search/No-license-filtering/0-128/1/cancel
- Chocobo background: https://www.cbr.com/final-fantasy-xiv-tips-tricks-strategies/
- Chocobo cartoonized: https://telegramhub.net/chocobo-sticker-pack/
- Chocobo with book: https://finalfantasy.fandom.com/wiki/Chocobo_series?file=CT_Chocobo.png
- Play now: http://code7-game.com/#home

- Chocobo gif: https://www.tumbral.com/blog/carpenter-cat-tomo-ffxiv

Sounds:

- Background music: https://www.youtube.com/watch?v=b58YxbOligI
- Chocobo sound: https://www.youtube.com/watch?v=Gwe6r5puMe0
- Game over music: https://www.youtube.com/watch?v=RHpn-o9n-cs&t=176s
- Explosion sound: https://www.youtube.com/watch?v=Y3y_Oq6M5mw

Snippets:

- Warning window: https://bootsnipp.com/snippets/QM68X