VR/AR and Unity exam

ALGOSUP 2022

I. Multiple-choice questionnaire

Answer the following questions. Write your answers in the *I-Questions answers.txt* file at the root of this repository. No negative points

1. Besides taking care of performances (optimizations), cite one other specificity/difference of a VR app compared to a standard flat screen 3D app

2. Which of these terms match the most the following definition: *The ability of a system to simultaneously capture its surrounding environment and locate itself within it*?
    A. HMD
    B. LIDAR
    C. SLAM

3. Which type of stereoscopy is implemented when a user is wearing anaglyph glasses (red and blue lenses)?
    A. Active stereoscopy (temporal separation)
    B. Passive stereoscopy (separation by filter)
    C. Autostereoscopy (spatial separation)

WORK TO DO:
1. Answer the 3 questions (3pts)

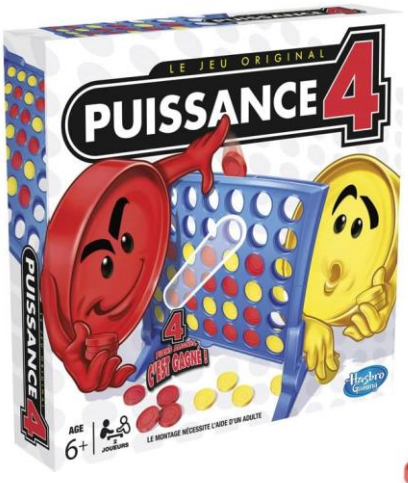
II. Connect4 game

In this section, you'll work on a Connect4 game (Puissance 4 in French). It is a two-player board game in which they take turns dropping colored pieces into a 7column, 6row grid, the objective is to be the first to form a horizontal, vertical or diagonal line of 4 of your own pieces.

DONE

WIP

NOT DONE



In our virtual version, the player is playing against an artificial intelligence (AI). It starts with a player turn. Each turn, a new piece is placed on top of the board above the middle column (3rd one). If it's player turn, he is able to move his new piece above neighboring columns using the left and right arrow keys and release it using the space bar. If it's AI turn, his actions are simulated. Once the piece reached its final position either at the bottom of the board or above another piece, the turn is finished and a new one starts.
The game ends as soon as the player or the AI has aligned 4 of his pieces in the board or if the board is full without winner (draw). At this point, a menu pops up and shows the result and a replay button. Pressing the replay button will hide the menu and restart a new game.

You will not start the project from scratch; you'll catch it partway through. Open the Unity project using Unity 2020.3.11f1. If you don't have this exact version, open it with any 2020 version. Open the *Scenes/II-Connect4* scene and take a quick look at how it is organized. The scene hierarchy is composed of:

- **`GameManager`**, a GameObject holding the turn-by-turn logic and input handling script (GameManager.cs)
- **`Board`**, the board itself composed of the board mesh, its colliders and **`Pieces`**, an empty GameObject that will be the container (the parent) of all instanced pieces (instanced prefabs)
- **`FinalUI`**, the user interface that shows up once the game has ended. It displays the result and a replay button

All scripts of this scene are located in *Scripts/II-Connect4/* folder.
The *Connect4Game.cs* script is handling the game logic (grid, win detection, etc…), you won't have to edit it.

For now and until your 4th mission is done, you can go to the next turn is only by pressing the tab key.

Here are your missions to finish the game:
A problem needs to be fixed: when releasing a piece, the latter is not dropping in a column but is stopped colliding with the board top instead.

For now, a piece can be released but doesn't seem to move from above a column to above another when pressing the arrow keys. Take a look at *GameManager.cs* and Piece.cs scripts. Complete the `MoveLeft` and `MoveRight` functions of the *Piece.cs* script to update the position of the concerned piece. You'll see that part of these functions has already been implemented.

All created pieces have the same color. Complete the `setOwner` function of the *Piece.cs* script so pieces get their color updated according to their owner (Player or AI). Use the materials referenced as `playerPieceMaterial` and `AIPieceMaterial` on the `GameManager` instance.

Now, let's detect when a piece reaches its final position so that we can directly start the next turn (instead of pressing tab). A piece has reached its final position when it collides either with the bottom of the board (first piece in the column) or with another piece (already present in the column). Complete the *Piece.cs* script to detect these collisions and when that happens call the appropriate function to start the next turn. Make sure to not go to next turn too early or multiple times (too late).
Then, remove the tab key press detection in *GameManager.cs* as you don't need it anymore.

Finally, create a pop animation on the final UI. The animation should last around half a second. It consists of modifying the size of the overall UI in 3 steps, starting so small that it is not visible and scaling up to a bit bigger than its normal size to finally be back to its normal size. This animation should play once, when the UI is shown.

Tips:
- All created pieces are instances of *Prefabs/Piece* prefab
- The `isReleased` and `hasReachedFinalPlace` parameters of the *Piece.cs* script are here for a reason
- When a previously hidden GameObject is shown, its attached Animator is reset (back to entry state)
- A tidy project is always easier to read (please use the existing Animations folder)

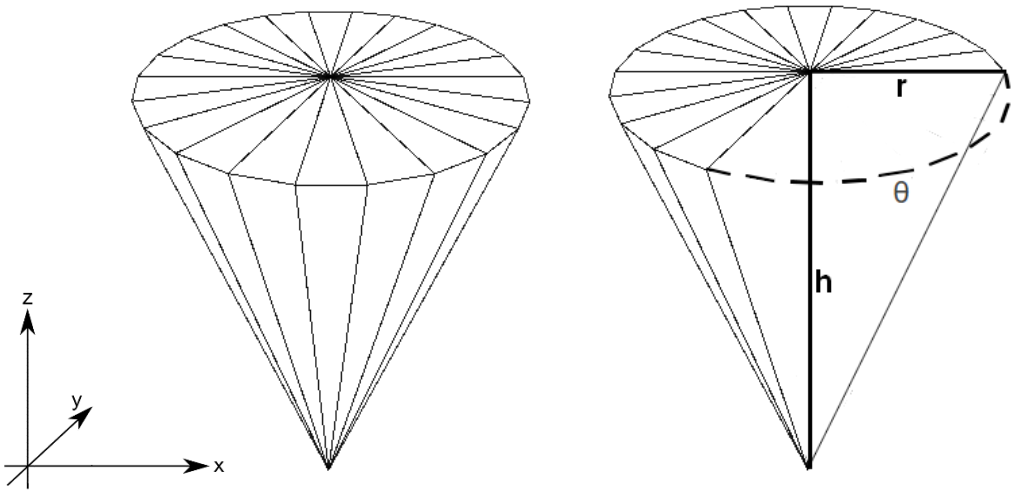
WORK TO DO:
1. Fix the issue of pieces not entering in columns (1pts)
2. Make the piece move between columns (2pts)
3. Change the color of piece according to owner (3pts)
4. Detect when a piece reaches its final position and call next turn (4pts)
5. Add a 'pop' animation to the final UI (2pts)

Look at *Connect4.mp4* at the root of this repository to see the expected result

III. Ice cream generation

In this section, the goal is to generate an ice cream cornet programmatically. To do so, we will create a new mesh and we will calculate the vertices, triangles, normal and UVs. Our cornet will be made of:

- A vertex at the bottom, z=0 (tip of the cornet)
- A circle of vertices at the top (z=height)
- A vertex at the center of this circle



The vertices along the circle can follow the following equation:

$x(\theta) = r\cos\theta$

$y(\theta) = r\sin\theta$

$z(\theta) = h$

$with\colon\ \theta \in [0, 2\pi)$

where

θ is the angle which make a full circle, so its value start and end at the same point,

r is the radius of the circle

h is the height of the cornet

Open the scene *Scenes/III-IceCreamGeneration* and take a look at the *IceCreamGenerator.cs* script located in *Scripts/III-IceCreamGeneration*. As you can see, most of the work is already done, triangles and UVs has been calculated following the above description. The last thing left to do is to calculate the vertices positions. Make sure to respect the chosen **`radius`**, **`height`** and resolution **`revolutionResolution`** which is the number of divisions of the circle.

As a bonus, you can vary the height of vertices in the circle part of the cornet to add some relief to it. One vertex out on two being higher (by the amount you judge look right) than the others

Tips:

- Trigonometry functions are available in the Mathf library (`Mathf.Sin()` for example) as well as PI value (`Mathf.PI`)
- Be careful, the up axis in Unity is Y and not Z.

WORK TO DO:

1. Complete the `GenerateIceCream` function in *IceCreamGenerator.cs* with vertices positions calculation (5pts)

Bonus. Continue editing the `GenerateIceCream` function to add some relief to the cornet (+1pts)

Here's the expected result, a tasty (but virtual) ice cream!

(with a bit of relief)