Section 2 Plan

1. Introduction to Section 2.

Short discussion of what students will learn in each lecture while showing parts of each lecture on the screen.

1. Creating a New Project and Importing Assets.

* Create a New empty 2d project.
* Drag our Game View to be side-by-side with the Scene View.
* Import assets from the folder Resources to Start 2048 Development by dragging everything into Project Tab in Unity.

1. Introduction to Unity User Interface (UI)

* UI is used to create: menus, conversation screens, inventories, HUDs for health and mana, progress bars, etc.
* We can access Unity UI from the context menu Create in the Hierarchy Tab.
* All UI elements are just images and text and scripts that hold them together.
* Canvas is an object that holds all our UI objects
* Canvas can be created manually from Create menu in Hierarchy or automatically with the first UI game object that we create.
* Create one of each UI Element and show that every game object consists of images and text.
* Rect tool is the best tool to work with UI in Unity.
* Enter the game mode and demonstrate clicking on buttons and switching toggles, etc…
* Save this scene and call it CanvasTest (talk about scenes in Unity, treat them as separate levels in the game or just pieces of the game that provide different experiences for the player)
* Create a folder in Assets called TestScenes and put our scene there.

1. Parenting and Sorting Order of Unity UI.

* Create a new scene.
* Parenting is an important concept that helps us keep our UI organized.
* Create 3 images and give them different colors to demonstrate parenting and sorting order.
* Create object as a child of an existing UI element by Right Clicking on that element and selecting any new UI component from the context menu.
* Child objects move together with their parent
* We can create empty game objects to group our UI.
* UI elements are sorted only according to their order in Hierarchy Tab.
* Elements that are on top of the list in Hierarchy are rendered first, and they appear in the bottom in our scene, covered by other UI elements.
* Elements that are last in our Hierarchy are rendered last, therefore appear on top of everything else.
* Save this scene and call it ParentingSortingTest, put it in our Test Scenes folder.

1. Assembling the Game Screen (Part One)

* Create a new scene and place it in the root of Assets folder.
* Drag our screenshot to the scene in front of the Main Camera. Create an empty Canvas. Double click on the Canvas.
* It might seem that our screenshot has disappeared, but we can still see it in the Game View. This is happening because of the difference in scale between canvas and all the game objects. (In Canvas 1 pixel on the canvas = 1 unit in our game = 100 pixels)
* This might seem confusing, but actually it is quite useful because the game objects like our characters, level details, etc won`t be a distraction while we work on the UI.
* Resize our screenshot to match the size of our canvas
* Create a Panel.
* Change its color and notice the difference. Remove default graphic from the panel.
* Create a game title, 2048.
* Talk about setting up the size of our text and Best fit option. Best fit adjusts the size of our text automatically between 2 values depending on the size of the bounding Rectangle for our text.

1. Assembling the Game Screen (Part Two)

* Add all the other texts and enable best fit for them (lower size = 3pt ).
* The text might not be sharp in the scene, but it will be sharp in our game (run the game).
* Create a New Game Button, change child text to say New Game.
* Create Score HUDs by parenting two Text objects to an Image. Create second HUD by duplicating the first one (Ctrl+D to duplicate game objects)
* Create a placeholder Image for our game field.

1. Adding Custom Styling to Parts Of Text Using RichText.

* First, I`m changing the color of all the Texts in the scene because by default they have darker color than the text on the sample screenshot.
* Why selecting a part of our text and hitting bold in Text settings won`t work.
* Add bold styling with HTML filters <b></b>
* Other effects that can be achieved are: italic <i></i>, size : <size=20> <size>.

1. Anchoring of Unity UI.

* Try to resize the panel and see that everything sticks to the center. This should be done often to check the quality of the setup.
* We want all the elements to stretch when we resize our panel.
* Talk about RectTransform. Looks different from an ordinary transform. It is just a rectangle.
* Each RectTransform has anchors – small triangles that can be moved to change its behavior.
* RectTransforms position and stretch/shrink behavior is defined by its position and anchoring preset relative to its parent. We can place anchors only on parent object`s rectangle and cannot place them out of its bounds.
* Default Anchoring preset – anchored to the Center.
* Demonstrate horizontal and vertical stretch anchoring presets on our GameField.
* RectTransform will not stretch if the anchors stick together.
* I will use the same anchoring preset for all the objects in our game. “The magic preset”.
* Place our UI where we want it to be, apply anchors to the corners.
* Try to resize Panel and check that everything now stretches.

1. Create a Grid of Tiles on the Game Field.

* Create an image that will represent one tile on our game field and pick a color for it.
* Placing and anchoring all the tiles manually is inefficient and may result in inconsistent spacing between the tiles. Hard to introduce changes.
* We can select our GameField, Add Component - Layout – GridLayoutGroup – this option places tiles automatically, but the biggest setback is that we have to manually select the size of each tile in pixels, and this size wont change when we resize the Game Field, therefore it will not look acceptable on different resolutions.
* Both of these options do not fit to make our game field stretch.

1. Making the Grid on the Game Field Resolution Independent.

* Remove GridLayoutGroup script from our GameField and replace it with the AutoGridLayout script from the Resources folder that we have imported in the beginning.
* Reconfigure AutoGridLayout: Check IsColumn and set column constraint to 4 columns, set appropriate padding and spacing.
* The tiles will stretch but we want our Game Field to maintain square shape for each resolution.
* Use search in AddComponent to find AspectRatioFitter.
* Create a parenting game object to constraint the area on the screen that our GameField wont overflow.
* Set AspectRatioFitter to FitParent option.

1. Add Numbers to All Tiles on the Game Field

* Select the first tile and create an image as a child of this Tile.
* Create a text as a child of our image, set alignment so that the digit appears in the center.
* Change Text to say 2, pick the colors.
* Delete all the other tiles and duplicate the tile with number 2 to fill the game field.
* Done – the screen for our game is complete.
* Recap what the students have learned in this section.
* In the next section we will switch to writing code for the game and making our game work: new tiles will appear, tiles will move on the game field, merge into bigger tiles, the score will count, etc…