**Section 3 Plan**

1. Introduction to Section 3
2. Reading Input from Keyboard

* Create 2 scripts – InputManager.cs and GameManager.cs
* In the InputManager get a reference to GameManager
* Read input from keyboard.
* Create an enum MoveDirection that will hold the values for our possible moves: Left, Right, Up and Down.
* Create a method Move() in the GameManager.cs script that will take a MoveDirection value.
* For test purposes add a Debug.log statement into the Move() method
* Attach all the scripts to our Canvas and check if everything works.

1. Storing Color Info for Different Tiles

* Create a script called TileStyleHolder to store all the information about the colors of tiles and numbers on the game field.
* Make this script a “Singleton” – create a public static variable that will hold reference to the only instance of this class and initialize it in Awake() method.
* Create a class TileStyle that will hold information about one tile state (tile number and colors for text and tile image).
* Mark this class as [System.Serializable] to make it visible in the inspector.
* Go back to Unity and populate TileStyleHolder with all the colors using the color picker or select colors manually.
* IMPORTANT – don’t forget to set alpha values for each color to 255 so that the tiles are not transparent.

1. Creating a Tile Script and Applying Visual Style to Tiles

* Create a new C# script called Tile
* Add a new using UnityEngine.UI; statement so that we can access our Text and Image.
* Get references to respected Text and Image components.
* Create method ApplyStyleFromHolder() that will take an entry from TileStyleHolder and apply it to this tile.
* Create another method called ApplyStyle that will take an integer value (2, 4, 8, …) and will change appearance of this tile accordingly.

1. Clear The Field When the Game Starts

* Create 2 new methods in the Tile script: SetVisible() and SetEmpty()
* Create a private number variable in the Tile class and a public Number property to access this variable.
* Put all the logic for changing the look of the tile into this property, so that just by saying: someTile.Number = 4 it will change all the visual properties of the tile.
* By convention setting Number to 0 will clear the tile.
* In Unity add Tile script to all the tiles.
* Go to GameManager and get references to all the tiles that we have in the scene.
* In Start() add a foreach operator to clear the field.

1. Store the Information About All the Tiles on the Game Field.

* Create an array called AllTiles
* Create 2 new integer fields in the Tile class , indX and indY that will represent indexes of each tile in the AllTiles array.
* Enter indexes for each tile manually in the inspector
* Populate the AllTiles array using indexes indX, indY for each Tile.

1. Randomly Generate New Tiles

* Create a list called EmptyTiles so that we know which tiles are available for random generation.
* When the game starts add all tiles to the EmptyTiles list because in the beginning of the game all the Tiles are empty.
* Create a method called Generate() to generate one new number 2 tile on an empty spot.
* Modify the method to have 1 out of 10 probability to spawn a number 4 tile.
* Create a way to test this method, so that when we press a key on the keyboard, a new tile will be generated.
* Test everything in Unity by entering play mode.

1. Main Game Mechanics of the 2048 Game

* This is the lecture when we view the slideshow to prepare for writing code that will move the tiles on the game field.
* Independent moves in each row for Left/Right moves
* Independent moves in each column for Up/Down moves
* Tile numeration
* What will matter in code: UpIndex and DownIndex moves.
* Elementary moves: move 1 tile to an empty spot and merge two neighbor tiles into one.
* Examples
* Tiles merge only once and the tile that is a product of merging that happened this turn can no longer participate in other merges on the same turn.

1. Provide Structure for Tiles in GameManager: Create Lists of Rows and Columns.

* Create rows and columns lists. These lists will hold Tile arrays that represent actual rows and columns on the game filed.
* When we create arrays using new statement we can assign all the elements in the array during initialization by listing them in brackets {}.

1. Create Methods for Shifting Tiles on the game field.

* Create a method MoveOneTileDownIndex(). It will track two consecutive tiles across one line on the game field (might be row or column) and if there is a move available – it will swap an empty tile with its neighbor number tile.
* Create MoveOneTileUpIndex() by changing some indexes and tracking direction.

1. Moving Tiles on the Game Field
2. Merging Tiles to Form Bigger Tiles

* Create a boolean variable in Tile class that will be called mergedThisTurn to prevent tiles from merging twice.
* Add merge logic to MakeOneMoveUpIndex() and MakeOneMoveDownIndex() methods.
* Create method to ResetMergedFlags.
* State the issues and bugs that we have: Empty tiles, bug with new tile appearing even after the move is not made.

1. Bug Fixes and Automatic Tile Generation.

* Add calls to Generate() method where we want our tiles to be generated.
* Fix the bug with empty tiles. Add method UpdateEmptyTiles.
* Fix the bug with generating a tile even if the move was not made. Introduce moveMade bool variable.

1. Making New Game Button Functional. And Create ScoreTracker Script.

* Create a method in GameManager that will reset the current scene in our game
* Add this method to the NewGame Button`s OnClick event
* Create a script ScoreTracker and make it a singleton

1. Score Tracking and Recording of High Scores

* PlayerPrefs is a place where we can save data about our game: scores, positions of players, etc…
* We can access entries in PlayerPrefs using string keys.
* Create a property that will control interaction with the ScoreTracker class. If current score is higher than current High Score saved in PlayerPrefs, value in PlayerPrefs should be updated.
* Modify GameManager to add the numbers on merged tiles to Score.
* Attach ScoreTracker to Canvas and test all the changes.

1. Making Game Over Panel

* Create a panel with a couple of texts and a button that will display information to the Player when the Game ends.

1. Game Over Conditions in Code.

* Create a method called CanMove that will check for available moves
* Write a GameOver() method that will display a message.
* Check for CanMove() in the end of Move() method. If there are no available moves, call GameOver method

1. Configure Game Over Panel

* Make Continue button do the same thing that the New Game Button does.

1. Display You Won Message.

* Create YouWon() method. Change the Message to YouWon! /n Your Score is:
* Call YouWon() method when two tiles get merged into a 2048 tile
* Make Continue Button have the same handler as a NewGameButton.
* Test everything (might test for 2048 tile by going to the inspector and changing to debug mode and changing numbers of tiles to 1024)