The Grid

I used the GameGrid class as the foundation for the game loop:

- Reads input from 3 input events (start touch, drag, end touch) and processes them to find which cells are involved in the movement.
- Applies the movement rules of the game (only between not empty adjacent cells)
- Queues up to 3 consecutive movement inputs that are executed one at a time. The more movements are queued, the faster the animation runs. This helps create a smoother interaction.
- When the queue empties, a scan through all cells verifies if the game is ended (no more movements possible), in which case the win condition is evaluated.
- Dispatches major gameplay events (win, lose, movement, levels completed)
- Uses the GameData scriptable object reference to get information about the game it has to play. It contains references to other scriptable objects that can be exchanged to modify game settings:
  - win condition,
  - animation that moves elements on the grid,
  - procedural level generator,
  - set of pre-serialized levels it has to go through.

  Note: Another useful setting would have been applying special functions after each move (like merging tiles in the 2048 version), but for time reasons I instead linked a behavior to the movement event (the TileMerger object in the scene).
- Switches between the two versions of the game at runtime.

Elements and Pools

I used "elements" as a generic term for ingredients and tiles as they are treated the same way on the Grid.

- Element is a struct containing both data and a reference to an object in the scene.
- The data part is a reference to a scriptable object; each element has its own.
- There is one generic prefab that can be used for each element.
- A pooling system saves memory allocations, creating new element game objects only when needed. Otherwise it just activates and deactivates existing ones.
- When needed, an element game object is provided by the pool and it gets its mesh and material applied from the data.

Level Generators

- They work both in the editor and at runtime.
- When in the editor, you can generate a new level in the inspector and a rough representation of the grid appears as a preview. If you like the result, you can save the level into a scriptable object or generate a new one (save path is fixed but can be easily modified).
- The sandwich algorithm starts with two adjacent pieces of bread (a necessary condition for a solvable puzzle) and branches off to neighboring cells until the set number of ingredients is reached.
- The 2048 algorithm starts with the total value of all tiles merged together in a cell. Then, iteratively a random occupied cell halves its value and sends the other half to a neighbor. It repeats until the set number of tiles is reached or no cell can be halved anymore (e.g. all occupied cells have a value of 2).

ScriptableObject Events

I used scriptable objects as events to be more versatile by linking them directly in the editor and reduce dependencies in the code.

- SOEventListener is a custom property that allows me to link the event in the inspector and manage the invocation method in code. The Component version instead uses a UnityEvent in order to be fully set in the editor.

Known issue: very fast movements can cause grid malfunction. Use the reset button.