Idle ecocity

Technical Design Document - ver. 0.1

Made by Valentina Tosto

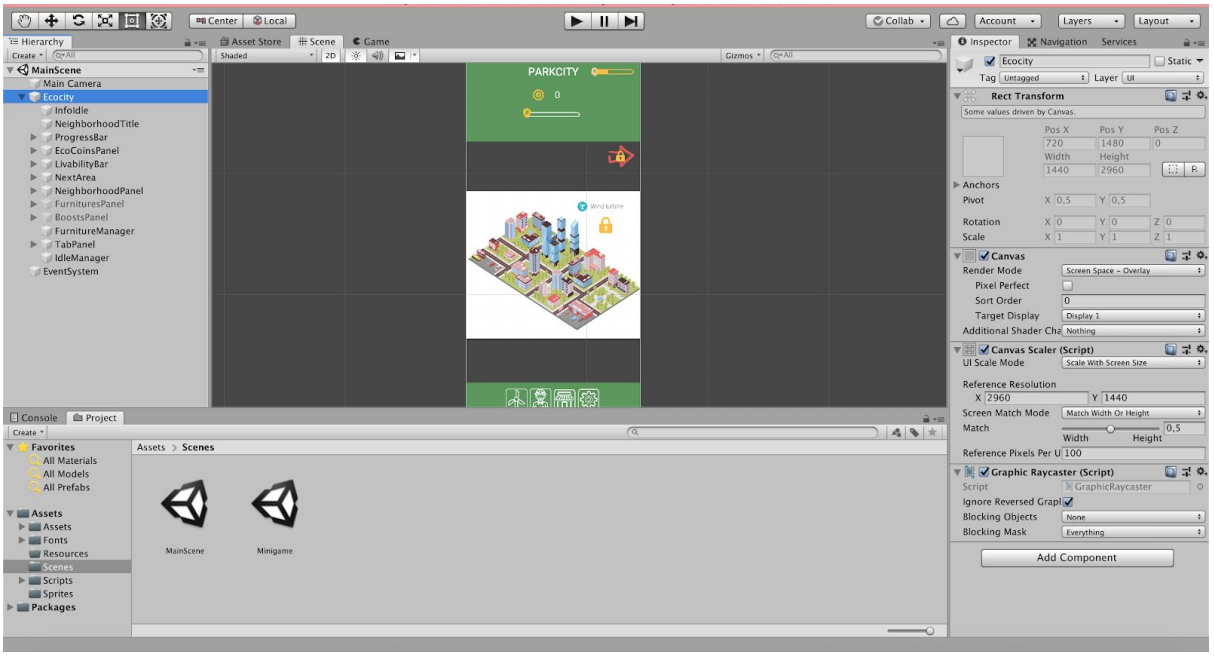
Table of contents

1. Scenes

The “Idle ecocity” game is structured in two scenes: **MainScene** and **Minigame**.

MainScene

This scene represents the idle component of the game. There is a canvas “Ecocity” that groups different GameObjects and Panels.
On the top are shown the progress bar of player (at present only a placeholder), the total eco-coins and the livability bar.
In the middle there is the first neighborhood, called “Parkcity”, where there’s the first furniture available (unlockable for free). Once the furniture (wind turbine) has been unlocked, the player can click on it, to start earning eco-coins. Moreover there is an arrow that is unlockable when the livability bar reaches a specific threshold.
On the bottom there is the tab panel, with furnitures, boosts, shop and settings. Currently only the furnitures and boosts tabs are available.



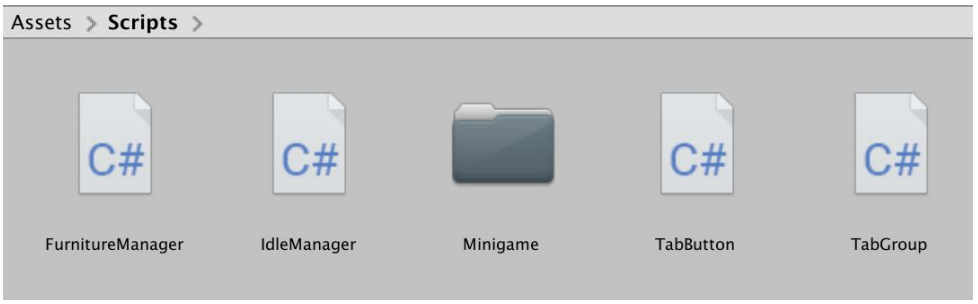
Minigame

This scene represents the minigame to unlock a new neighborhood.
On the top there is an arrow to return to the first area and a panel with items (solar panels) to position in the grid.
In the middle there is the real minigame, structured in a grid 4 x 4 (the first level), with three houses that need to be powered.
On the bottom there is a panel with obstacles to position in the grid, to avoid that the connections of solar panel cross each other, and a button to confirm the minigame solution.
Currently there are the drag and drop and the connections of solar panels handler.

2. Scripts

MainScene

The MainScene is composed by four scripts, to manage the idle, the furnitures and the tabs.



In the **IdleManager** are loaded and saved the variables, as eco-coins, coins earned by a furniture and livability bar, so that it's not necessary to restart the game.
Then, in this script are managed the update of eco-coins value and of livability bar.
There is four methods, in addition to those relating to update, saving and loading data:

- *ClickToIncreaseCoins()* needs to increase coins manually, earned by a specific furniture present in the neighborhood
- *AutoPlayFurniture()* needs to increase coins automatically, based on coins earned per second, unlockable after the first upgrade of a furniture
- *ChangeNumber(double amount)* needs to change the value of a number, when it's a thousand or a million, to reduce its length
- *GoToMinigame(string minigame)* needs to go to next area, when the player click on relative button (after that the livability bar has exceeded a certain threshold and the padlock disappears).

In the **FurnitureManager** are loaded and saved the variables, as coins per sec, upgrade level and upgrade cost of a furniture.
Then, in this script are managed all the upgrade of a furniture, hence the purchase and the increase of levels and coins per sec, based on an increment upgrade that, after a definite number of upgrade levels, enhances by a percentage.
Moreover, in this script is managed the flow of the furniture's progression bar present in the neighborhood, that increases related to upgrade levels.
Finally, each five upgrade levels, the livability bar grows by 10%.
There is one method, in addition to those relating to start, update, saving and loading data:

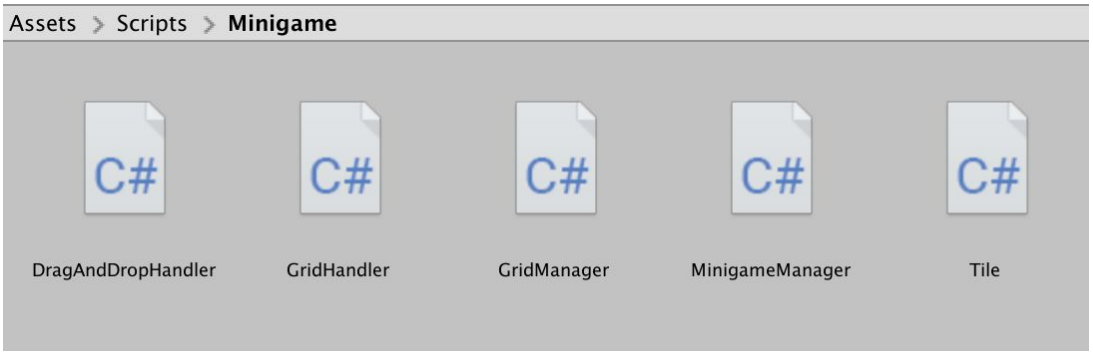
- *BuyUpgradeFurniture()* needs to upgrade a furniture, paying the indicated cost, that change according to level number of upgrade. The upgrade bar of furniture increases at each specific level range; when it fills up, the coins per sec enhances by a percentage.

The **TabButton** and the **TabGroup** are linked between them, because they manage the tab panel. TabButton is related to the tab itself, there is the addition of it in tabs group and the handler of tab selection/deselection. Moreover, in this script it's present an interface implementation of the event system "IPointerClickHandler", that detect the tab click.
In TabGroup there is the handler of tab addition and selection in two methods:

- *AddTabsToList(TabButton tab)* needs to add each tab in a list
- *OnTabSelected(TabButton tab)* needs to select the right tab and deselect the previous tab or the tab itself, if this was already activated.

Minigame

The Minigame scene is composed by five scripts, to handle the grid and drag and drop of items.



The **DragAndDropHandler** manages the events system about the drag of solar panels and obstacles. There are four methods, in addition to those relating to start and awake to initialize variables:

- *OnBeginDrag(PointerEventData eventData)* needs to detect the begin of item drag. if the item was in a grid tile, in this method are deactivated the item connections in the grid
- *OnDrag(PointerEventData eventData)* needs to set dynamically the new position that the item assumes on every frame
- *OnEndDrag(PointerEventData eventData)* needs to detect the end of item drag and the anchor to initial position, if the item isn't dropped in the grid
- *DeactivateConnections()* needs to disable all item connections, based on grid structure, when begins the item drag.

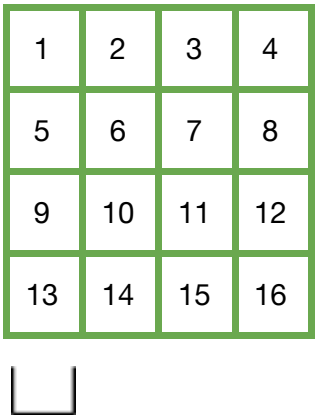
In the **GridHandler** there is the drop handler of an item. If the item is dropped in a valid position (a tile of the grid), it's anchored to center of tile and are shown the vertical and horizontal connections of solar panel.
There are two methods, in addition to start method:

- *OnDrop(PointerEventData eventData)*, this is the interface implementation to handle the event system about the drop. Here the item is anchored to a valid position and get the right tile reference. Then, the solar panel connections are activated, based on the tile position in the grid

- *ActivateConnections(Tile tileReference, DragAndDropHandler item)* needs to activate the right connections relating to tile reference. According to the tile id, the vertical nodes of upper and lower tiles (if are present) are activated and the horizontal nodes of the left and right tiles (if are present).

The **GridManager** manages the grid creation, required to identify the tile position when an item is moved to it. There are two methods, in addition to start:

- *AddTileToList(Tile tile)* needs to add each tile to a tile list
- *CreateGrid()* needs to create a 4x4 grid, adding four rows lists, where each list contains a tile of each row.



Rows list

In the **MinigameManager** it's managed only the back button, to return to the previous area and exit the minigame. There is only one method:

- *BackToMain(string main)* needs to handle the change of scene.

The **Tile** identifies the GameObject related to tile. This script needs to detect if it's necessary to activate or deactivate its connections. There is only one method:

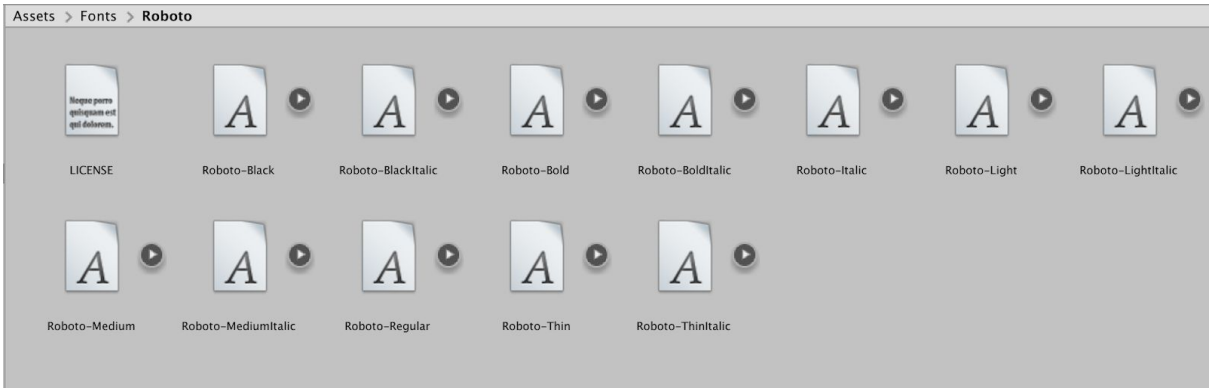
- *IsActived(bool activated, string type)* needs to activate the connections (that are the tile children), where type specifies vertical or horizontal node.

3. Sprites

This is the folder that contains all sprites (icons and images) used for the UI.

4. Fonts

This is the folder that contains fonts used for the UI.



5. Assets

This is the folder with additional UI assets, used for the bars and buttons, taken from the Unity store.