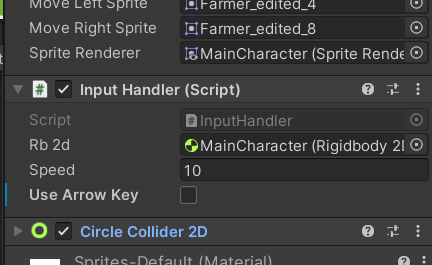
FarmVille

# Design Pattern

## {Huyền} [Behavioral Pattern] Command Pattern: *Main Character Movement - Keyboard*

* Code trc khi có Command:
  + 
* => Sử dụng Command Pattern:
  + Có 2 cách di chuyển:
    - Dùng phím mũi tên
    - Dùng WASD
  + => Ng dùng có thể tự do chuyển đổi giữa 2 cách di chuyển = cách sửa “Use Arrow Key”
    - 
* **Implementation**:
  + Command: abstract class cho các Command implement
  + MoveRight, MoveLeft, MoveUp, MoveDown, Stop, DoNothing: các Command implement, tương ứng với từng hành động di chuyển
  + InputHandler: Sử dụng Command Pattern:
    - Định nghĩa các Object Command
    - -> Tùy theo người dùng lựa chọn sử dụng cách di chuyển nào mà khởi tạo Command tương ứng cho từng Object
    - -> Mỗi khi ng dùng bấm 1 Button -> Gọi đến method thực thi hành động của Command tương ứng
* *Sử dụng kết hợp với State Design Pattern*

## {Giang} [Behavioral Pattern] State Design Pattern: *Main Character Animation Transition*

* Sử dụng Pattern để quản lý Animation của Nhân vật chính:
  + Nhân vật sẽ có những State di chuyển khác nhau: lên, xuống, trái, phải
  + Mỗi State lại có hành động khác nhau: thể hiện Sprite tương ứng
  + => Sử dụng State Design Pattern để quản lý và thay đổi State di chuyển cho Nhân vật mỗi khi Nhân vật có hướng di chuyển mới -> từ đó thể hiện Sprite tương ứng phù hợp
* **Implementation**:
  + IMoveState: interface cho các States implement
  + MoveLeftState, MoveRightState, MoveUpState, MoveDownState: các States khác nhau của Animation, implement hành động (chuyển sprite) tương ứng cho state đó
  + MoveStateContext: dùng để lưu trữ State hiện tại của GameObject và chuyển state
  + MoveController: điều khiển các State = cách sử dụng Context
  + InputHandler: (Client – Sử dụng Pattern) gọi đến Controller và qua đó gọi đến từng method chuyển state khác nhau *(Kết hợp với Command Pattern -> Code bên trong từng Command)*

## {Giang} [Behavioral Pattern] State Design Pattern: *Farm*

* Sử dụng Pattern để quản lý những State khác nhau của 1 Ô đất (ruộng):
  + 1 Ô đất sẽ có các State khác nhau:
    - Chưa trồng
    - Đã trồng
    - Đã tưới nước
    - Đã chín
  + Mỗi State lại có thể hiện Sprite khác nhau và làm dc hành động khác nhau: thứ tự chuyển State luôn phải là: Chưa trồng -> (Trồng) -> Đã trồng -> (Tưới nước) -> Đã tưới nước -> (Đợi -> Chín) -> Đã chín -> (Thu hoạch) -> Chưa trồng
    - Nếu chuyển State ko theo thứ tự này thì hành động sẽ ko hợp lệ (ko cho phép)
  + => Sử dụng Pattern để quản lý và thay đổi các State, các Class khác sẽ điều khiển Ô đất thông qua việc sử dụng StateController:
    - Người chơi: trồng; tưới nước; thu hoạch
    - Mưa: tưới nước
    - Ô đất: làm cho cây chín (sau 1 khoảng thời gian)
* **Implementation**:
  + Plant: 1 enum để lưu trữ các loại cây khác nhau ng dùng trồng
  + IFarmState: interface cho các States implement
  + FarmHarvestState, FarmRipeState, FarmSeedState, FarmWaterState: các State khác nhau của Farm, chuyển FarmPlot sang State tương ứng nếu kiểm tra thấy điều kiện phù hợp
  + FarmStateContext: sử dụng để lưu trữ State hiện tại và chuyển State
    - Truyền Controller cho từng State để nó thay đổi các thuộc tính
  + FarmController: điều khiển các State = cách sử dụng Context
    - Khi cây chín -> tùy thuộc vào loại cây ng dùng trồng (truyền vào Controller ở method chuyển state: *FarmRipe()*) -> tạo hiển thị khác nhau
    - Khi thu hoạch (và bán) -> tùy thuộc vào loại cây ng dùng trồng -> nhận về 1 lượng tiền khác nhau:
      * *Cà rốt:* ***50$***
      * *Bí ngô:* ***70$***
      * *Gạo:* ***90$***
      * *Hoa hướng dương:* ***100$***
  + FarmPlot: (Client – Sử dụng Pattern)
    - gọi đến Controller và qua đó gọi từng method chuyển State khác nhau
    - tính thời gian chín cho cây -> khi cây chín thì sẽ tự gọi chuyển state *FarmRipe()*
    - lưu trữ loại cây đc trồng:
      * Farmer truyền vào qua method *FarmSeed()*
      * Dùng để truyền vào FarmController khi cây chín qua method *FarmRipe()*
  + Farmer: (Client – Sử dụng FarmPlot)
    - Điều khiển FarmPlot khi người dùng nhấn các button
      * Thông qua việc:
        + Tìm FarmPlot gần vị trí Main Character nhất
        + Gọi đến các Method của nó
    - Hướng dẫn sử dụng:
      * *Trồng: nhấn* ***J***
      * *Tưới nước: Nhấn* ***K***
      * *Thu hoạch: Nhấn* ***L***

## {Giang} [Creational Pattern] Object Pool Pattern: (Combine with Flyweight Pattern)

* Khi tạo 1 cơn mưa sẽ phải khởi tạo 1 lượng lớn các Object giọt mưa r lại hủy chúng -> sử dụng Object Pool để quản lý
* **Implementation**:
  + RainDrop: (*PooledObject*) Script cho object để làm Pool
    - Khi đc khởi tạo sẽ tồn tại (rơi tự do, vì có RigidBody2D) trong 2 giây rồi destroy
    - Dùng *Timer* để đếm thời gian
  + RainController: (Client using Pool) Script để khởi tạo và sử dụng Pool
    - Sử dụng ObjectPool của UnityEngine.Pool (Unity 2021) để tạo Pool
    - Mỗi 10 giây sẽ tạo ra 1 cơn mưa => khởi tạo ra 1 lượng lớn các object RainDrop tại các vị trí ngẫu nhiên trên Map
    - Mỗi khi mưa -> (Sử dụng State Design Pattern) Làm cho tất cả các ô đất được tưới nước

# Algorithm

## {Giang} Dijkstra Shortest Path Algorithm: *Main Character Movement – Mouse Click*

* Sử dụng Shortest Path Algorithm để tìm đường đi ngắn nhất trong 1 Đồ thị với các Đỉnh là các điểm trên Map và các Cạnh là các đường đi trong Map => Khi người dùng Click chuột vào 1 điểm thì sẽ tìm Đỉnh gần vị trí đó nhất và tìm đường đi ngắn nhất từ vị trí Nhân vật đang đứng đến điểm đó, rồi dần dần di chuyển đến
* **Implementation**:
  + ShortestPathClient: xác định điểm bắt đầu và đích đến khi người dùng Click chuột, gọi đến Traveler để thực hiện thuật toán
  + Traveler: Thực hiện thuật toán để tìm đường đi ngắn nhất, sử dụng 1 SortedLinkedList gồm các SearchNode để lưu trữ đường đi đó, rồi di chuyển Nhân vật dần dần đến từng Đỉnh trong Đồ thị
  + GraphBuilder: tạo ra 1 Đồ thị dựa vào các gameObject trên Map làm Đỉnh và các Cạnh được định nghĩa trong code
  + EdgeRenderer: vẽ ra các Cạnh đã được định nghĩa sau khi Build Graph (dùng để Debug chứ không hiện lên Map)
* **Algorithmic Complexity**: **O(v2)**
  + V: số cạnh của Graph

# Data Structure

## {Giang} Undirected Weighted Graph: *Used in: [Dijkstra Shortest Path Algorithm]*

* Đồ thị gồm các Đỉnh là các điểm trên Map và các Cạnh là đường đi từ đỉnh này đến đỉnh khác => Khi người dùng click vào 1 điểm thì sẽ tìm đường đi từ đỉnh ở vị trí đang đứng đến đỉnh ở điểm Click
* **Implementation**:
  + GraphNode: 1 Đỉnh của Graph, trong đó gồm List các Đỉnh kề nó (có cạnh nối nhau) và độ dài cạnh đến những Đỉnh đó
  + Graph: Lưu trữ List các GraphNode và các method để thêm/xóa Node và thêm/xóa Cạnh

## {Giang} Sorted Linked List: *Used in: [Dijkstra Shortest Path Algorithm]*

* 1 Linked List kiểu Generic dùng để sử dụng trong Shortest Path, có thể sắp xếp theo thứ tự
* **Implementation**:
  + SortedLinkedList<T>: kế thừa LinkedList<T> của C#
    - *Add*(): thêm 1 item vào list
    - ***Reposition***(): sắp xếp 1 item vào vị trí đúng của nó trong List

## {Giang} Queue: *Used in: [User Cash Flow]*

* 1 Queue chứa các dữ liệu thuộc enum Plant để lưu trữ các loại hạt giống mà người dùng đã mua:
  + Khi người dùng mua hạt giống => Enqueue
  + Khi người dùng lấy hạt giống ra để trồng => Dequeue

=> Đảm bảo luôn lấy hạt giống cũ nhất ra để trồng

=> Tránh chuyện hạt giống để lâu bị hỏng

* **Implementation**:
  + Queue: class để định nghĩa 1 Queue kiểu Generic
    - Enqueue: thêm vào đầu list
    - Dequeue: lấy ra từ cuối list
  + Farmer: (Client) Sử dụng Queue để lưu trữ 1 list các Seed người dùng mua
    - Nhân vật có lượng tiền khởi điểm là 50$, có thể dùng lượng tiền này để mua hạt giống

Nếu ko đủ tiền để mua sẽ hiện *thông báo* và hủy giao dịch

Hạt giống đc mua sẽ đc *Enqueue*

* + - Hướng dẫn sử dụng:
      * *Tiền khởi điểm:* ***50$***
      * *Mua Hạt giống:*
        + *Cà rốt: Nhấn* ***1*** *(Giá:* ***10$****)*
        + *Bí ngô: Nhấn* ***2*** *(Giá:* ***20$****)*
        + *Lúa: Nhấn* ***3*** *(Giá:* ***30$****)*
        + *Hoa hướng dương: Nhấn* ***4*** *(Giá:* ***40$****)*
    - Khi người dùng trồng cây (nhấn J) thì sẽ lấy 1 hạt giống từ Queue ra để trồng (*Dequeue*)
      * Hiện thông báo nếu ko có hạt giống nào

# File I/O

## {Giang} Store Data to File:

* Lưu trữ dữ liệu chơi của người dùng vào File -> khi người dùng tắt game và chơi lại thì tiếp tục tiến độ
  + Lưu trữ:
    - Lượng tiền người dùng kiếm dc
    - Các Hạt giống người dùng đã mua
  + => Khi người dùng thoát game -> Lưu dữ liệu vào File  
    Khi người dùng mở lại game (vào scene GamePlay) -> Đọc dữ liệu từ File
* **Implementation**:
  + UserData: class định nghĩa dữ liệu để ghi vào file, gồm tiền và các hạt giống
  + JsonHandler: class để thực hiện các method đọc/ghi file với kiểu dữ liệu trên; cho các class khác gọi đến các method của nó để thực hiện business
  + ChangeScene: (*Client*) khi QuitGameAndSaveData() -> gọi đến JsonHandler để ghi dữ liệu Tiền và các Hạt giống của người dùng hiện đang có vào File
  + Farmer: (Client) khi khởi tạo object (bắt đầu game – Awake) -> gọi đến JsonHandler để đọc dữ liệu từ file
    - Nếu ko có File hoặc dữ liệu đọc ra không hợp lệ (Lượng Tiền và Hạt giống khiến người dùng không thể chơi game) thì sẽ không lấy dữ liệu đó mà khởi tạo dữ liệu mặc định

# Others

## {Giang} Background & Map Creation:

[***https://learn.unity.com/tutorial/introduction-to-tilemaps***](https://learn.unity.com/tutorial/introduction-to-tilemaps)

**Tilemap - Guide**:

* 1. What is Tilemap?
* 2. Installing the Tilemap Editor
  + Ko cần
* 3. Create a Tilemap
  + **Hierarchy**: 2D Objects -> Tilemap -> Rectangular
* 4. About the Tilemap system
  + Ở phần Grid có thể chỉnh các thuộc tính:
    - Cell Size
    - Call Gap
    - Call Layout
    - Call Swizzle
* 5. Configuring the Tilemap
  + *Tilemap Component*
  + *Tilemap Renderer Component*
* 6. Exploring the **Tile Palette**
  + *Window -> 2D -> Tile Palete*
  + Tools:
    - Selection
    - Move
    - Bursh
    - Fill Selection
    - Tile Sampler
    - Eraser
    - Fill
* 7. Using the Tile Palette
  + Create New Tile Palette
* 8. Using Tilemap Collider 2D
  + Tilemap Renderer -> Add Component: **Tilemap Collider 2D**

**Tilemap – Implementation:**

* **Tilemap**:
  + **TileMap\_Background**: vẽ những Object nằm ở Background
    - Gồm: Đường đi
    - **Sorting Layer**: *Background*
  + **TileMap\_Platformer:** vẽ những Object trên Map, có **Collider**
    - Gồm: Ô đất, Tường bao quanh Map
    - **Sorting Layer**: *Platformer*
* **Tile Palettes:**
  + Folder **Palettes**: Lưu trữ các Palette khác nhau
    - *Background\_Road\_FarmPlots*
    - *Background\_Road\_FarmPlots\_2*
  + Folder **Tiles**: Lưu trữ các Tiles trong Palette
* **Grid**:
  + *Cell Size: 0.6* (để phù hợp với kích cỡ của Palette)

## {Giang} User Cash Flow:

* Dòng tiền của nhân vật:
  + Khởi điểm: *50$*
    - Số tiền này dùng để làm vốn, từ đó người dùng sẽ mua hạt giống -> trồng cây -> bán để kiếm thêm tiền
  + [Queue] Mua hạt giống:
    - Dùng tiền để mua các loại hạt giống khác nhau (trừ tiền sau khi mua)
    - Nếu ko đủ tiền để mua sẽ hiện thông báo và hủy giao dịch
    - Hạt giống đc mua sẽ Add vào Queue
  + [Queue] Trồng: lấy hạt giống từ Queue ra
  + [State Design Pattern - Farm] *Thu hoạch (và bán): Nhận lại 1 lượng tiền:*
    - *Cà rốt:* ***50$***
    - *Bí ngô:* ***70$***
    - *Gạo:* ***90$***
    - *Hoa hướng dương:* ***100$***

## {Huyền} Other Scenes:

* [StartMenu]:
  + Guide: to Scene [Guide]
  + Start: to Scene [GamePlay]
  + Quit: quit game (with Application.Quit())
* [Guide]:
  + Back: to Scene [StartMenu]

## {Huyền} Slides – Presentation