**TRƯỜNG ĐẠI HỌC CẦN THƠ**

**KHOA CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG**

**BỘ MÔN CÔNG NGHỆ PHẦN MỀM**

**NIÊN LUẬN CƠ SỞ NGÀNH KTPM**

**GAME 2048**

|  |  |
| --- | --- |
| **Sinh viên thực hiện**  **Hồ Phú Giàu**  **MSSV: B2012082** | **Cán bộ hướng dẫn**  **Ông Thị Mỹ Linh** |

**HỌC KỲ 1, 2023-2024**

**ĐÁNH GIÁ CỦA GIÁO VIÊN HƯỚNG DẪN**

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

.....................................................................................................................

# I.MÔ TẢ GAME 2048

1.1 Mô tả game 2048

**Game 2048** là một trò chơi điện tử thuộc thể loại puzzle đơn giản và dễ thao tác, được tạo ra bởi Gabriele Cirulli vào năm 2014.

Về luật chơi, game 2048 khác với game xếp hình, cần kết hợp các ô số bằng cách di chuyển chúng nhập vào nhau đến khi tạo nên được số 2048.

Game lần đầu tiên được viết ra bằng JavaScript và CSS và phát hành ngày 09-03-2014. Game sau khi được đăng tải thì nhận được hơn 4 triệu lượt chơi trong gần 1 tuần. Sau đó, game 2048 đã được viết ra thêm trên nhiều nền tảng như Android, iOS, Windows,... và được biến tướng theo nhiều kiểu chơi và quy luật chơi khác nhau.

Trong báo cáo, em xin sử dụng ngôn ngữ C# và công cụ Unity để phát triển trò chơi 2048.

1.2 Giới thiệu công cụ Unity

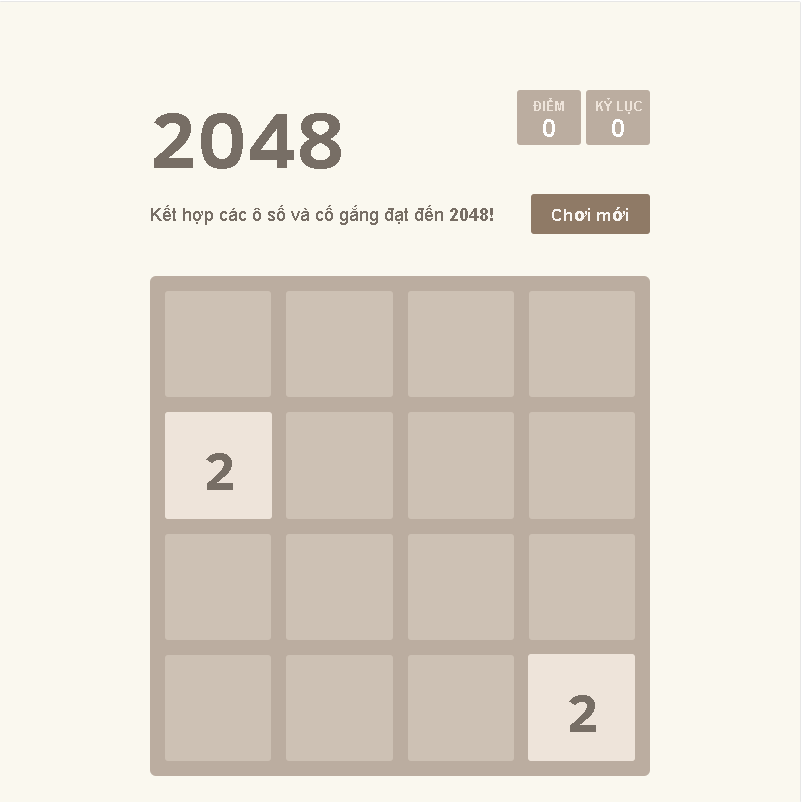
**Unity** là một môi trường phát triển trò chơi (game development environment) mạnh mẽ và phổ biến, được sử dụng rộng rãi trong ngành công nghiệp trò chơi để tạo ra các trò chơi 2D và 3D đa nền tảng. Dưới đây là mô tả sơ lược về chức năng của Unity:

* + Giao diện người dùng:
    - Giao diện của Unity dựa trên nguyên tắc kéo và thả (drag-and-drop) giúp người dùng dễ dàng làm việc với các tài nguyên và đối tượng trong trò chơi.
    - Có các cửa sổ quan trọng như Scene (hiển thị cảnh trò chơi), Hierarchy (liệt kê đối tượng trong cảnh), Inspector (hiển thị và chỉnh sửa thông tin đối tượng), và Project (quản lý tài nguyên).
  + Đa nền tảng:
    - Unity hỗ trợ phát triển trò chơi cho nhiều nền tảng như Windows, macOS, iOS, Android, WebGL, PlayStation, Xbox, Nintendo Switch, và nhiều nền tảng khác.
  + Ngôn ngữ lập trình:
    - Unity hỗ trợ nhiều ngôn ngữ lập trình, như C#, JavaScript, và Boo. Tuy nhiên, trong thời gian gần đây, C# trở thành ngôn ngữ lập trình phổ biến nhất trong Unity.
  + Hỗ trợ đồ họa và âm thanh:
    - Unity tích hợp mạnh mẽ cho việc xử lý đồ họa 2D và 3D với hỗ trợ cho các định dạng file như PNG, JPG, FBX, OBJ, v.v.
    - Hỗ trợ âm thanh bao gồm các tệp âm thanh, âm thanh nén, và hiệu ứng âm thanh.
  + Physics Engine:
    - Unity có một hệ thống vật lý tích hợp giúp mô phỏng các hiệu ứng vật lý như va chạm, trọng lực, và động lực.

# CHƯƠNG 2: PHỔ BIẾN LUẬT CHƠI

## 2.1 Luật chơi

Trò chơi 2048 diễn ra trên một bảng lưới có kích thước 4x4. Trong mỗi nước đi, người chơi sử dụng các phím mũi tên để di chuyển các ô vuông. Các ô vuông sẽ di chuyển theo hướng được chỉ định (lên, xuống, trái, phải). Sau mỗi nước đi, một ô vuông mới có giá trị 2 hoặc 4 sẽ xuất hiện ngẫu nhiên trong một ô trống trên bảng.



H2.1.Giao diện khi bắt đầu trò chơi

Để dễ phân biệt, các ô vuông có giá trị khác nhau sẽ có màu sắc khác nhau. Bảng điểm ở góc trên bên phải của màn hình thể hiện điểm số của người chơi. Ban đầu, điểm số bằng 0. Khi hai ô vuông kết hợp, điểm số của người chơi tăng lên bằng giá trị của ô vuông mới tạo ra. Kết quả cao nhất mà người chơi từng đạt được được hiển thị bên cạnh điểm số hiện tại.

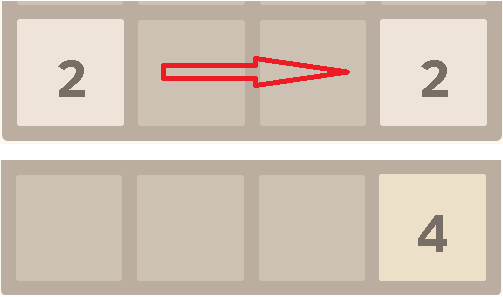
      

H2.3.Các khối khác màu trong game.

## 2.2 Di chuyển , kết hợp ô

Các ô vuông di chuyển cho đến khi chúng chạm vào biên của bảng hoặc chạm vào một ô vuông khác. Nếu hai ô vuông có cùng giá trị va chạm vào nhau, chúng sẽ kết hợp lại thành một ô vuông mới có giá trị là tổng giá trị của hai ô vuông ban đầu (ví dụ: nếu hai ô vuông 2 gặp nhau, chúng sẽ tạo ra một ô vuông 4).



H2.2.Di chuyển các ô qua phải, 2 ô 2 va chạm vào nhau tạo thành ô 4.

Lưu ý : *Khối vuông kết quả sau khi kết hợp không thể tiếp tục kết hợp với khối vuông khác trong cùng một nước đi.*

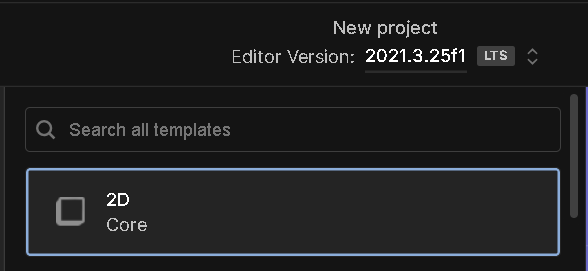
## 2.3 Kết thúc trò chơi

Trò chơi kết thúc khi người chơi tạo ra một ô vuông có giá trị 2048, và người chơi chiến thắng. Nếu không còn nước đi hợp lệ (không còn ô trống trên bảng và không có cặp ô vuông cạnh nhau có cùng giá trị), trò chơi sẽ kết thúc.

# CHƯƠNG 3: THIẾT KẾ GIAO DIỆN

## 3.1 Tạo dự án

Tạo một project 2D core bằng phiên bản hiện tại của Unity trong máy tính.



## 3.2 Tạo tài nguyên ( Asset)

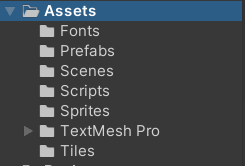
Sử dụng các công cụ và tài nguyên có sẵn trong Unity để thiết lập các hình ảnh, âm thanh và các thành phần khác cần thiết cho trò chơi

3.2.1 Title.png

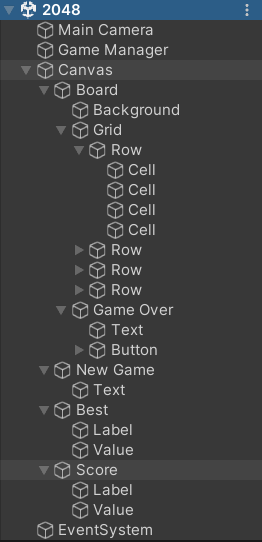
Tạo title.png để có thể sử dụng nó như một bản đồ tileset trong Unity. Bằng cách này, bạn có thể tạo nhanh chóng và dễ dàng xây dựng môi trường game mà không cần phải vẽ từng chi tiết một. Mỗi tile đại diện cho một phần của môi trường game hoặc một phần của cấu trúc thế giới trong trò chơi.

3.2.2 Các ổ chứa

Tại folder Assets chứa các folder con ( Fonts gồm các font chữ đã được thêm vào có dạng .ttf, Sprites chứa Tile.png, Scenes : 2048, TextMeshPro, Tile).



Tại 2048 (Main Camera, Canvas, Board, Background, Grid, Game Manager)



3.2.3 Chỉnh sửa và bổ sung các thuộc tính

Tại Scenes có SamepleScene : đổi tên thành 2048

Thiết lập lại asset title.png :

* sửa pixel per units : 3
* Generate Physics Shape : no
* Format : RGBA 32bit
* Max size : 512
* Sprite Editor : Border (L,R,B,T) : 16,16,16,16

Tại TextMeshPro :

* chọn Import TMP Essential Resources : Import hết các thành phần.
* Tạo các TextMeshPro của các file .ttf trong Fonts => sinh ra SDF

Tại Main Camera :

* Background : FAF8EF

Tạo UI Canvas , khi tạo sinh ra thêm EventSystem:

UI Scale Mode :

* + - * + Scale with Screen Size
        + Reference Resolution (X,Y) : 1280, 720
        + Match : 0.5

## 3.3 Thiết lập khung

Xác định kích thước và định dạng của khung trò chơi

Tạo Object tên Board :

* Width, Height : 500,500
* Tạo UI Image tên Background
  + - * Color : BBADA0
      * Source Image : Title.png
      * Anchor Presets : All Stretch
      * Left,Top,Right,Bottom : 0,0,0,0
* Tạo Object tên Grid
  + - * Anchor Presets : All Stretch
      * Left,Top,Right,Bottom : 16,16,16,16
      * Tạo Object tên Row
        1. Tạo Object tên Cell
    1. Tạo UI Image tên Cell
    2. Source Image : Title.png
    3. Pixels Per Unit Multiplier : 3
    4. Color : CDC1B4
       - 1. Thực hiện tạo lần lượt thêm 3 UI Image Cell cùng tên giống với Cell ở trên.
* Thực hiện tạo lần lượt thêm 3 Object Row cùng tên giống với Row ở trên
* Chọn tất cả các Row :

1. Add Component -> Horizontal Layout Group
2. Spacing : 16
3. Control Child Size : Width yes, Height yes

* Chọn Grid :

1. Add Component -> Vertical Layout Group
2. Spacing : 16
3. Control Child Size : Width yes, Height yes

Remove tất cả VerticalLayout Group của Grid

Remove tất cả Horizontal Layout Group của Row

=> tránh lỗi vặt sau này

## 3.4 Thiết lập ô

Tạo các ô vuông hoặc hình chữ nhật đại diện cho các ô trong lưới của game 2048

Tại Canvas tạo UI Image tên là Tile

* Width, Height : 105,105
* Source Image : Title.png
* Pixels Per Unit Multiplier : 3
* Tạo UI Text-TextMeshPro tên là Text :

Căn giữa văn bản bằng Alignment

Font Asset: Bold SDF

Auto Size : yes

Anchor Presets : All Stretch

Left,Top,Right,Bottom : 16,16,16,16

Sau khi thiết lập xong Tile,tạo Folder mới tên Prefabs

Đưa Tile vào trong Folder Prefabs

Xóa Tile trong Canvas

## 3.5 Thiết kế khối

Thiết kế các khối ban đầu với các giá trị và màu sắc khác nhau để bắt đầu trò chơi

Tạo Folder tên Scripts

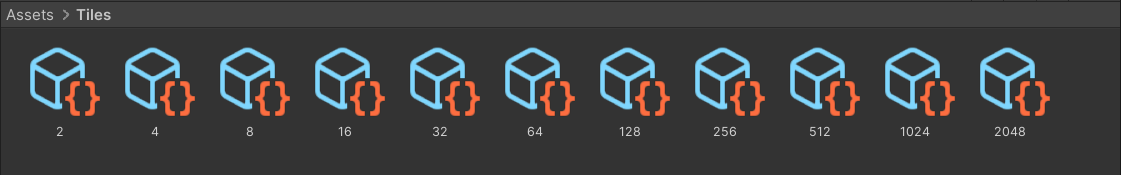
Tạo C# Script tên TileState

Tại TileState.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **using** **UnityEngine**;  [CreateAssetMenu(menuName = "Tile State")]  **public** **class** **TileState**: ScriptableObject {  **public** Color backgroundColor;  **public** Color textColor;  **public** **int** number;  } |

Tạo Folder tên Tiles : trong Tiles tạo mới các Tile State :

|  |  |  |  |
| --- | --- | --- | --- |
| Tile State | Background Color | Text Color | Number |
| 2 | EEE4DA | 776E65 | 2 |
| 4 | EEE1C9 | 776E65 | 4 |
| 8 | F3B27A | F9F6F2 | 8 |
| 16 | F69664 | 776E65 | 16 |
| 32 | F77C5F | 776E65 | 32 |
| 64 | F75F3B | 776E65 | 64 |
| 128 | EDD073 | 776E65 | 128 |
| 256 | EDCC62 | 776E65 | 256 |
| 512 | EDC950 | 776E65 | 512 |
| 1024 | EDC53F | 776E65 | 1024 |
| 2048 | EDC22E | 776E65 | 2048 |



Lưu ý : Chỉnh giá Trị A của Background Color lên tối đa (255) để khi tạo các khối sẽ không bị độ trong của màu làm mờ đi.

# CHƯƠNG 4: KHUNG CHƯƠNG TRÌNH

## 4.6 Thiết kế trạng thái của khối

Định nghĩa các trạng thái, hành vi và ý nghĩa của khối

Tại Folder Scripts tạo C# Script tên Tile

Đồng thời tạo C# Script tên TileCell, TileRow, và TileGrid

* Sử dụng hành động kéo-thả để đưa Script vào các Object
* Đưa TileGrid.cs vào Object Grid
* Đưa TileRow.cs vào tất cả các Object tên Row
* Đưa TileCell.cs vào tất cả các Object tên Cell
* Đưa Tile.cs vào UI Image tên Tile

Tại TileCell.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **using** **UnityEngine**;  **public** **class** **TileCell**: MonoBehaviour {  **public** Vector2Int coordinates {  **get**;  **set**;  }  **public** Tile tile {  **get**;  **set**;  }  **public** **bool** empty => tile == **null**;  **public** **bool** occupied => tile != **null**;  } |

Tại TileRow.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **using** **UnityEngine**;  **public** **class** **TileRow**: MonoBehaviour {  **public** TileCell[] cells {  **get**;  **private** **set**;  }  **private** **void** **Awake**() {  cells = GetComponentsInChildren < TileCell > ();  }  } |

Tại TileGrid.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | **using** **UnityEngine**;  **public** **class** **TileGrid**: MonoBehaviour {  **public** TileRow[] rows {  **get**;  **private** **set**;  }  **public** TileCell[] cells {  **get**;  **private** **set**;  }  **public** **int** size => cells.Length;  **public** **int** height => rows.Length;  **public** **int** width => size / height;  **private** **void** **Awake**() {  rows = GetComponentsInChildren < TileRow > ();  cells = GetComponentsInChildren < TileCell > ();  **for** (**int** i = **0**; i < cells.Length; i++) {  cells[i].coordinates = **new** Vector2Int(i % width, i / width);  }  }  } |

Tại Tile.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **using** **UnityEngine**;  **public** **class** **Tile**: MonoBehaviour {  **public** TileState state {  **get**;  **private** **set**;  }  **public** TileCell cell {  **get**;  **private** **set**;  }  **public** **int** number {  **get**;  **private** **set**;  }  } |

## 4.7 Xử lý trên các khối

Viết mã để xử lý các tương tác của người chơi với các khối, bao gồm việc di chuyển và kết hợp chúng

## 4.8 Tạo ngẫu nhiên các khối

Tạo các hàm để sinh ra các khối ngẫu nhiên khi bắt đầu trò chơi hoặc khi người chơi thực hiện hành động như di chuyển , new game.

Trong Folder tạo C# Script tên TileBoard

Sử dụng hành động kéo-thả để đưa TileBoard.cs vào Object Board

Tại TileBoard.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **using** **System.Collections.Generic**;  **using** **UnityEngine**;  **public** **class** **TileBoard**: MonoBehaviour {  **public** Tile tilePrefab;  **private** TileGrid grid;  **private** List < Tile > tiles;  **public** TileState[] tileStates;  } |

Khi thiết lập xong TileBoard.cs , tại Object Tile hiện có Script Tile Prefab, và Tile States List

Kéo-thả UI Image Tile vào Tile Prefab

Kéo-thả các Tile State tương ứng với các trường Element n trong List Tile States

Có thể xóa các Element dư thừa trong List

Thêm vào TileBoard các hàm:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **private** **void** **Awake**() {  grid = GetComponentInChildren < TileGrid > ();  tiles = **new** List < Tile > (**16**);  }  **public** **void** **CreateTile**() {  Tile tile = Instantiate(tilePrefab, grid.transform);  tile.SetState(tileStates[**0**]);  tile.Spawn(grid.GetRandomEmptyCell());  tiles.Add(tile);  } |

Thêm vào Tile thư viện và các hàm:

|  |  |
| --- | --- |
| 1  2 | **using** **UnityEngine.UI**;  **using** **TMPro**; |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **private** Image background;  **private** TextMeshProUGUI text;  **private** **void** **Awake**() {  background = GetComponent < Image > ();  text = GetComponentInChildren < TextMeshProUGUI > ();  }  **public** **void** **SetState**(TileState state) {  **this**.state = state;  background.color = state.backgroundColor;  text.color = state.textColor;  text.text = state.number.ToString();  }  **public** **void** **Spawn**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = cell;  **this**.cell.tile = **this**;  transform.position = cell.transform.position;  } |

Thêm hàm vào TileGrid :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **public** TileCell **GetRandomEmptyCell**() {  **int** index = Random.Range(**0**, cells.Length);  **int** startingIndex = index;  **while** (cells[index].occupied) {  index++;  **if** (index >= cells.Length) {  index = **0**;  }  // all cells are occupied  **if** (index == startingIndex) {  **return** **null**;  }  }  **return** cells[index];  } |

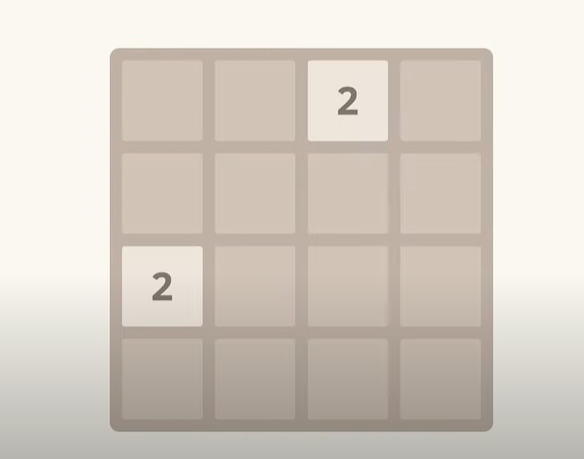
Khi bổ sung hàm Start() vào TileGrid.cs và Start() vào TileBoard.cs, lúc này ta thấy game chạy theo đúng hướng và có kết quả tạo được 2 ô cùng giá trị tại bất kì địa điểm nào (2 hàm Start được sử dụng tại đây chỉ nhất thời giúp cho việc giải thích một phần phía trên rằng code đã chạy theo đúng tiến độ và sẽ dời hàm Start vào các vị trí khác thích hợp hơn).

Start() tại TileGrid.cs:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | private void Start() {  for (int y = 0; y < row.Lenght; y++) {  for (int x = 0; x < rows[y].cells.Lenght; x++) {  row[y].cells[x].coordinates = new Vector2Int(x, y);  }  } |

Start() tại TileBoard.cs:

|  |  |
| --- | --- |
| 1  2  3  4 | private void Start() {  CreateTile();  CreateTile();  } |



Sau các bước thiết lập cho khối, ấn chạy chương trình sẽ thấy trong bảng được tạo 2 khối xuất hiện ngẫu nhiên trong bảng có cùng giá trị là 2: (Bổ xung phần code bị xóa vào đây để thể hiện rằng khi có start game sẽ chạy theo hướng này)

## 4.9 Di chuyển các khối

Viết mã xử lý hành vi di chuyển của các khối khi người chơi nhấn các phím di chuyển

Tại TileBoard thêm vào các hàm :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | **private** **void** **Update**() {  //Can bo sung them  **if** (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpArrow)) {  Move(Vector2Int.up, **0**, **1**, **1**, **1**);  } **else** **if** (Input.GetKeyDown(KeyCode.A) || Input.GetKeyDown(KeyCode.LeftArrow)) {  Move(Vector2Int.left, **1**, **1**, **0**, **1**);  } **else** **if** (Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.DownArrow)) {  Move(Vector2Int.down, **0**, **1**, grid.height - **2**, -**1**);  } **else** **if** (Input.GetKeyDown(KeyCode.D) || Input.GetKeyDown(KeyCode.RightArrow)) {  Move(Vector2Int.right, grid.width - **2**, -**1**, **0**, **1**);  }  }  **private** **void** **Move**(Vector2Int direction, **int** startX, **int** incrementX, **int** startY, **int** incrementY) {  //Can bo sung them  **for** (**int** x = startX; x >= **0** && x < grid.width; x += incrementX) {  **for** (**int** y = startY; y >= **0** && y < grid.height; y += incrementY) {  TileCell cell = grid.GetCell(x, y);  **if** (cell.occupied) {  //MoveTile chua hoan thanh  MoveTile(cell.tile, direction);  }  }  }  //Can bo sung them  }  **private** **void** **MoveTile**(Tile tile, Vector2Int direction) {  TileCell newCell = **null**;  TileCell adjacent = grid.GetAdjacentCell(tile.cell, direction);  **while** (adjacent != **null**) {  **if** (adjacent.occupied) {  //Xu ly va cham  **break**;  }  newCell = adjacent;  adjacent = grid.GetAdjacentCell(adjacent, direction);  }  **if** (newCell != **null**) {  tile.MoveTo(newCell);  //Can bo sung them  }  //Can bo sung them  } |

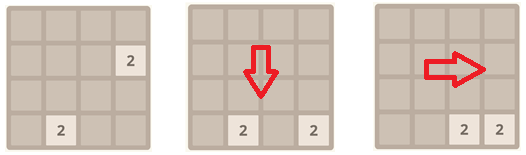
Tại TileGrid thêm vào các hàm :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **public** TileCell **GetCell**(**int** x, **int** y) {  **if** (x >= **0** && x < width && y >= **0** && y < height) {  **return** rows[y].cells[x];  } **else** {  **return** **null**;  }  }  **public** TileCell **GetCell**(Vector2Int coordinates) {  **return** **GetCell**(coordinates.x, coordinates.y);  }  **public** TileCell **GetAdjacentCell**(TileCell cell, Vector2Int direction) {  Vector2Int coordinates = cell.coordinates;  coordinates.x += direction.x;  coordinates.y -= direction.y;  **return** **GetCell**(coordinates);  } |

Tại Tile thêm vào :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **void** **MoveTo**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = cell;  **this**.cell.tile = **this**;  transform.position = cell.transform.position  } |

Như vậy, sau khi hoàn tất các bước, chúng ta có thể di chuyển các khối ô cùng dồn về 1 phía



## 4.10 Tạo độ trễ

Xác định và thiết lập độ trễ để tạo ra trải nghiệm game mượt mà hơn

Tại Tile chỉnh sửa và thêm vào hàm :

+ Chỉnh sửa :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **void** **MoveTo**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = cell;  **this**.cell.tile = **this**;    StartCoroutine(Animate(cell.transform.position));  } |

+ Thêm vào :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **private** IEnumerator **Animate**(Vector3 to) {  **float** elapsed = **0** f;  **float** duration = **0.1** f;  Vector3 **from** = transform.position;  **while** (elapsed < duration) {  transform.position = Vector3.Lerp(**from**, to, elapsed / duration);  elapsed += Time.deltaTime;  **yield**  **return** **null**;  }  transform.position = to;  //Can bo sung them  } |

Tại TileBoard thêm và sửa :

+ Thêm vào:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **private** **bool** waiting;  **private** IEnumerator **WaitForChanges**() {  waiting = **true**;  **yield**  **return** **new** **WaitForSeconds**(**0.1** f);  waiting = **false**;  //1  //2  } |

+ Tại MoveTile() chuyển từ void sang bool:

|  |  |
| --- | --- |
| 1 | **private** **bool** **MoveTile**(Tile tile, Vector2Int direction) |

Bổ sung vào đoạn cuối MoveTile() tại dòng 51 và 54

|  |  |
| --- | --- |
| 1  2  3  4  5 | **if** (newCell != **null**) {  tile.MoveTo(newCell);  **return** **true**;  }  **return** **false**; |

+Tại Move() :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **private** **void** **Move**(Vector2Int direction, **int** startX, **int** incrementX, **int** startY, **int** incrementY) {  **bool** changed = **false**;  **for** (**int** x = startX; x >= **0** && x < grid.width; x += incrementX) {  **for** (**int** y = startY; y >= **0** && y < grid.height; y += incrementY) {  TileCell cell = grid.GetCell(x, y);  **if** (cell.occupied) {  changed |= MoveTile(cell.tile, direction);  }  }  }  **if** (changed) {  StartCoroutine(WaitForChanges());  }  } |

+ Chỉnh sửa hàm Update : thêm if để đợi một khoảng thời gian sau khi nhận lệnh di chuyển:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **private** **void** **Update**() {  **if** (!waiting) {  **if** (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpArrow)) {  Move(Vector2Int.up, **0**, **1**, **1**, **1**);  } **else** **if** (Input.GetKeyDown(KeyCode.A) || Input.GetKeyDown(KeyCode.LeftArrow)) {  Move(Vector2Int.left, **1**, **1**, **0**, **1**);  } **else** **if** (Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.DownArrow)) {  Move(Vector2Int.down, **0**, **1**, grid.height - **2**, -**1**);  } **else** **if** (Input.GetKeyDown(KeyCode.D) || Input.GetKeyDown(KeyCode.RightArrow)) {  Move(Vector2Int.right, grid.width - **2**, -**1**, **0**, **1**);  }  }  } |

Như vậy, đã tạo được sự mượt mà khi các ô di chuyển và tạo một khoảng độ trễ cho hành động nhất định để khi di chuyển các khối với nhau sẽ có cảm giác dễ chịu.

## 4.11 Xử lý va chạm giữa các khối

Viết mã xử lí kết quả khi các ô di chuyển và kết hợp lại với nhau

Để xử lý va chạm chúng ta cần thêm vào hàm và bổ sung vào đoạn mã code của MoveTile() :

+Thêm vài hàm vào TileBoard.cs:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **private** **bool** **CanMerge**(Tile a, Tile b) {  **return** a.state == b.state; //  }  **private** **void** **MergeTiles**(Tile a, Tile b) {  tiles.Remove(a);  a.Merge(b.cell);  **int** index = Mathf.Clamp(IndexOf(b.state) + **1**, **0**, tileStates.Length - **1**);  TileState newState = tileStates[index];  b.SetState(newState);  //  }  **private** **int** **IndexOf**(TileState state) {  **for** (**int** i = **0**; i < tileStates.Length; i++) {  **if** (state == tileStates[i]) {  **return** i;  }  }  **return** -**1**;  } |

+ Thêm vào MoveTile() :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | **private** **bool** **MoveTile**(Tile tile, Vector2Int direction) {  TileCell newCell = **null**;  TileCell adjacent = grid.GetAdjacentCell(tile.cell, direction);  **while** (adjacent != **null**) {  **if** (adjacent.occupied) {  **if** (CanMerge(tile, adjacent.tile)) {  MergeTiles(tile, adjacent.tile);  **return** **true**;  }  **break**;  }  newCell = adjacent;  adjacent = grid.GetAdjacentCell(adjacent, direction);  }  **if** (newCell != **null**) {  tile.MoveTo(newCell);  **return** **true**;  }  **return** **false**;  } |

+Thêm hàm vào Tile.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **void** **Merge**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = **null**;  //  StartCoroutine(Animate(cell.transform.position, **true**));  } |

Tại Animate() thêm vào :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **private** IEnumerator **Animate**(Vector3 to, **bool** merging) {  **float** elapsed = **0** f;  **float** duration = **0.1** f;  Vector3 **from** = transform.position;  **while** (elapsed < duration) {  transform.position = Vector3.Lerp(**from**, to, elapsed / duration);  elapsed += Time.deltaTime;  **yield**  **return** **null**;  }  transform.position = to;  **if** (merging) {  Destroy(gameObject);  }  } |

+Chỉnh sửa lại một lần nữa ở MoveTo() :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **void** **MoveTo**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = cell;  **this**.cell.tile = **this**;  StartCoroutine(Animate(cell.transform.position, **false**));  } |

+Thêm vào WaitForChanges việc khi các khối kết hợp , trong lúc đó sẽ sinh thêm khối khác :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | **private** IEnumerator **WaitForChanges**() {  waiting = **true**;  **yield**  **return** **new** **WaitForSeconds**(**0.1** f);  waiting = **false**;  **if** (tiles.Count != grid.size) {  CreateTile();  }  //  } |

Hoàn thành xong các bước, hiện tại game đã có thể kết hợp các khối lại với nhau để tạo thành một khối có giá trị cao hơn

\*Xử lý lỗi va chạm không đúng :

- Khi bảng hiện tại đang có các khối mang giá trị liền kề nhau là 2,2,4. Việc di chuyển để kết hợp chúng lại tạo thành một khối mang giá trị là 8. Điều này là không hợp lý đối với quy luật vận hành của trò chơi. Chúng ta cần sửa lại thành khi kết hợp các khối sẽ tạo thành 2 ô mang giá trị lần lượt là 4,4.

Tại Tile.cs thêm vào :

|  |  |
| --- | --- |
| 1 | **public** **bool** locked { **get**; **set**; } |

Tại hàm Merge() thêm vào:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **void** **Merge**(TileCell cell) {  **if** (**this**.cell != **null**) {  **this**.cell.tile = **null**;  }  **this**.cell = **null**;  cell.tile.locked = **true**; //fix lai loi 224 -> 8  StartCoroutine(Animate(cell.transform.position, **true**));  } |

Tại TileBoard.cs chỉnh sửa CanMerge() :

|  |  |
| --- | --- |
| 1  2  3 | **private** **bool** **CanMerge**(Tile a, Tile b) {  **return** a.state == b.state && !b.locked;  } |

Thêm Vào WaitForChange():

|  |  |
| --- | --- |
| 1  2  3 | **foreach**(**var** tile **in** tiles) {  tile.locked = **false**;  } |

## 4.12 Thiết lập thắng thua

Xác định các điều kiện thắng thua, và viết mã để kiểm tra xem người chơi đã chiến thắng hoặc thất bại hay chưa

Tạo một C# Script GameManager.cs

Tạo một Emty Object Game Manager

Thêm ClearBoard vào TileBoard.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | **public** GameManager gameManager;  **public** **void** **ClearBoard**() {  **foreach**(**var** cell **in** grid.cells) {  cell.tile = **null**;  }  **foreach**(**var** tile **in** tiles) {  Destroy(tile.gameObject);  }  tiles.Clear();  }  **public** **bool** **CheckForGameOver**() {  **if** (tiles.Count != grid.size) {  **return** **false**;  }  **foreach**(**var** tile **in** tiles) {  TileCell up = grid.GetAdjacentCell(tile.cell, Vector2Int.up);  TileCell down = grid.GetAdjacentCell(tile.cell, Vector2Int.down);  TileCell left = grid.GetAdjacentCell(tile.cell, Vector2Int.left);  TileCell right = grid.GetAdjacentCell(tile.cell, Vector2Int.right);  **if** (up != **null** && CanMerge(tile, up.tile)) {  **return** **false**;  }  **if** (down != **null** && CanMerge(tile, down.tile)) {  **return** **false**;  }  **if** (left != **null** && CanMerge(tile, left.tile)) {  **return** **false**;  }  **if** (right != **null** && CanMerge(tile, right.tile)) {  **return** **false**;  }  }  **return** **true**;  } |

Bổ sung đoạn code cho WaitForChange():

|  |  |
| --- | --- |
| 1  2  3 | **if** (CheckForGameOver()) {  gameManager.GameOver();  } |

Tại GameManager.cs :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | **using** **UnityEngine**;  **public** **class** **GameManager**: MonoBehaviour {  **public** TileBoard board;  **public** CanvasGroup gameOver;  **private** **void** **Start**() {  NewGame();  }  **public** **void** **NewGame**() {  gameOver.alpha = **0** f;  gameOver.interactable = **false**;  board.ClearBoard();  board.CreateTile();  board.CreateTile();  board.enabled = **true**;  }  **public** **void** **GameOver**() {  board.enabled = **false**;  gameOver.interactable = **true**;  StartCoroutine(Fade(gameOver, **1** f, **1** f));  }  **private** IEnumerator **Fade**(CanvasGroup canvasGroup, **float** to, **float** delay = **0** f) {  **yield**  **return** **new** **WaitForSeconds**(delay);  **float** elapsed = **0** f;  **float** duration = **0.5** f;  **float** **from** = canvasGroup.alpha;  **while** (elapsed < duration) {  canvasGroup.alpha = Mathf.Lerp(**from**, to, elapsed / duration);  elapsed += Time.deltaTime;  **yield**  **return** **null**;  }  canvasGroup.alpha = to;  } |

Tại Object

Kéo-thả Object Board vào Script Board của Game Manager

kéo-thả Object Game Manager vào Script Game Manager của Board

Tạo UI Image tại Object Board tiêu đề là Game Over

Source Image : Tile.png

Anchor Presets : full Stretch

Left,Right,Top,Bottom : 0,0,0,0

Color : EEE4DA ,và giảm giá trị của A-Color xuống nhằm tạo độ trong suốt của UI.

Tạo TextMeshPro có Text Input :Game Over

Font Asset : Bold SDF

Color : 776E65

Font Size :60

Tạo UI Button-TextMeshPro

Source Image : Tile.png Pixel Per Unit : 3

One Click () : có Object là Game Manager

Funtion: GameManager.NewGame

Text Input :Try Again

Kéo-thả Object Game Over vào Script Game Over của Object Game Manager

Như vậy đã thiết lập xong mục tiêu thắng thua của trò chơi!

## 4.13 Tạo bảng điểm

Thiết lập hệ thống điểm và viết mã để cập nhật và hiển thị điểm số của người chơi khi họ chơi game.

Trong Object Canvas:

Tạo thêm một bảng UI Button New Game mới có giá trị giống với UI Button đã tạo trước đó

Tạo UI Image tên Best (Source Image : Tile.png ,Add Component : Vertical Layout Group

Tạo 2 UI TextMeshPro + Label : TextInput là Best

+ Value : TextInput là 0

Tạo UI Image chứa nội dung giống với Best tên là Score

+ Label : TextInput là Score

+ Value : TextInput là 0

Trong Object Game Manager :

Kéo-thả TextMeshPro Value của 2 Object Score và Best lần lượt vào Script tương ứng với tên gọi

Tại GameManager.cs , thêm vào đó :

|  |  |
| --- | --- |
| 1 | **using** **TMPro**; |

thêm vào NewGame():

|  |  |
| --- | --- |
| 1  2 | SetScore(**0**);  hiscoreText.text = LoadHiscore().ToString(); |

và thêm trong class:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | **public** TextMeshProUGUI scoreText;  **public** TextMeshProUGUI hiscoreText;  **private** **int** score;  **public** **void** **IncreaseScore**(**int** points) {  SetScore(score + points);  }  **private** **void** **SetScore**(**int** score) {  **this**.score = score;  scoreText.text = score.ToString();  SaveHiscore();  }  **private** **void** **SaveHiscore**() {  **int** hiscore = LoadHiscore();  **if** (score > hiscore) {  PlayerPrefs.SetInt("hiscore", score);  }  }  **private** **int** **LoadHiscore**() {  **return** PlayerPrefs.GetInt("hiscore", **0**);  } |

Tại TileBoard.cs thêm vào Merge():

|  |  |
| --- | --- |
| 1 | gameManager.IncreaseScore(newState.number); |

Như vậy , ta có thể lưu điểm hiện tại và điểm cao nhất vào trong game.



# CHƯƠNG 7: BIỂU ĐỒ TRÌNH TỰ GAME

# CHƯƠNG 8: TỔNG KẾT

## 8.1 Kết quả đạt được:

8.1.1Giải quyết vấn đề:

a.Tạo Game 2048:

* Đã phát triển thành công trò chơi 2048 bằng công cụ Unity, cung cấp trải nghiệm chơi game tương tự như phiên bản gốc trên di động và web.

b.Đối tượng người chơi:

* Hướng đến đối tượng người chơi yêu thích trò chơi logic và thách thức trí tuệ.

8.1.2 Thu hoạch chuyên môn:

Nâng cao kỹ năng lập trình:

* Có cơ hội tiếp xúc và ứng dụng nhiều kiến thức lập trình trong môi trường game.

Hiểu biết về giải thuật:

* Phát triển kỹ năng thiết kế giải thuật thông qua việc xây dựng thuật toán quản lý và di chuyển các ô số trong trò chơi.

Kinh nghiệm:

Phát triển game 2D:

* Đã có cơ hội áp dụng kiến thức về phát triển game 2D trong Unity.

Quản lý tài nguyên:

* Học cách quản lý tài nguyên hình ảnh, âm thanh và các thành phần khác trong dự án game.

## 8.2 Ưu điểm:

Unity làm tăng hiệu suất phát triển:

* Sử dụng Unity giúp tăng tốc độ phát triển và giảm độ phức tạp của quá trình lập trình game.

Dễ mở rộng và tùy chỉnh:

* Unity cung cấp cơ hội mở rộng dự án và tùy chỉnh trò chơi một cách linh hoạt.

## 8.3 Hạn chế và nguyên nhân:

Thiếu độ phức tạp và sự đa dạng trong trò chơi:

* Game 2048 có thể được coi là đơn giản, không mang lại độ phức tạp lớn và sự đa dạng trong trải nghiệm chơi game.

Thách thức về sáng tạo:

* Phát triển một trò chơi 2048 có thể giới hạn khả năng sáng tạo so với việc tạo ra một trò chơi độc đáo và mới lạ.

## 8.4 Hướng phát triển:

Mở rộng tính năng:

* Nâng cấp trò chơi bằng cách thêm tính năng mới, mode chơi, hoặc thậm chí tạo ra phiên bản nâng cao với đồ họa và hiệu ứng đặc sắc.

Đa dạng hóa trải nghiệm:

* Phát triển các trò chơi mới, hoặc tích hợp các yếu tố sáng tạo để tạo ra trải nghiệm đa dạng hơn cho người chơi.

Học hỏi từ người chơi:

* Thu thập phản hồi từ người chơi để hiểu rõ hơn về mong đợi và cải thiện chất lượng của trò chơi.

Kỹ năng xã hội và quảng bá:

* Phát triển kỹ năng quảng bá và tiếp thị để thu hút người chơi và tăng sự nhận thức về trò chơi 2048.

## Tóm lại, việc lập trình trò chơi 2048 bằng Unity mang lại nhiều thuận lợi trong việc học hỏi và phát triển kỹ năng lập trình game, nhưng đòi hỏi sự sáng tạo và nỗ lực liên tục để tạo ra trải nghiệm chơi game độc đáo và hấp dẫn.