Document

1. Introduction

This folder contains the code used in Canopy Quest for generating random terrain. The primary class responsible for this task is the `TreeDungeonGenerator`.

2. AbstractDungeonGenerator

The `AbstractDungeonGenerator` class serves as a base class for dungeon generation in Unity. It is designed to be extended by subclasses that implement specific procedural generation logic.

2.1 `GenerateDungeon`

- **Description:** Clears the tile map visualizer and triggers the procedural generation process.

2.2 `RunProceduralGeneration` (Abstract)

- **Description:** Subclasses must implement this method to define specific procedural generation logic.

3. ProveduralGenerationAlgorithms

The `ProceduralGenerationAlgorithms` static class provides several procedural generation algorithms commonly used in generating random walks, corridors, and employing binary space partitioning.

3.1 `SimpleRandomWalk`

```
public static HashSet<Vector2Int> SimpleRandomWalk(Vector2Int startPosition, int
walkLength)
```

- **Description**: Generates a simple random walk path starting from the given position.
- **Parameters**:
  - `startPosition`: The starting position of the walk.
  - `walkLength`: The number of steps in the random walk.
- **Returns:** A `HashSet<Vector2Int>` representing the path.

3.2 `RandomWalkCorridor`

```
public static List<Vector2Int> RandomWalkCorridor(Vector2Int startPosition, int
corridorLength)
```

- **Description**: Generates a corridor-like path using random walk logic.
- **Parameters**:
  - `startPosition`: The starting position of the corridor.
  - `corridorLength`: The length of the corridor.
- **Returns:** A `List<Vector2Int>` representing the corridor path.

3.3 `BinarySpacePartitioning`

```
public static List<BoundsInt> BinarySpacePartitioning(BoundsInt spaceToSplit,int
minWidth,int minHeight)
```

- **Description**: Applies binary space partitioning to divide a space into rooms.
- **Parameters**:
  - `spaceToSplit`: The initial space to partition.
  - `minWidth`: The minimum width of a room.
  - `minHeight`: The minimum height of a room.
- **Returns:** A `List<BoundsInt>` representing the resulting rooms.

3.4 `SplitHorizontally`

```
private static void SplitHorizontaly(int minWidth, Queue<BoundsInt> roomsQueue,
BoundsInt room)
```

- **Description**: Splits a room horizontally during binary space partitioning.
- **Parameters**:
  - `minWidth`: The minimum width of a room.
  - `roomsQueue`: The queue of rooms being processed.
  - `room`: The room to be split.

3.5 `SplitVertically`

```
private static void SplitVerticaly(int minHeight, Queue<BoundsInt> roomsQueue,
BoundsInt room)
```

- **Description**: Splits a room vertically during binary space partitioning.
- **Parameters**:
  - `minHeight`: The minimum height of a room.
  - `roomsQueue`: The queue of rooms being processed.
  - `room`: The room to be split.

3.6 `Direction2D`

- **Description**: The `Direction2D` static class provides lists of cardinal, diagonal, and eight directions in 2D space. It also includes a method to get a random direction.
- **Properties**
  - `cardinalDirectionsList`: A list of cardinal directions.
  - `diagonalDirectionsList`: A list of diagonal directions.
  - `eightDirectionList`: A list of eight directions.

4. SimpleRandomWalkDungeonGenerator

The `SimpleRandomWalkDungeonGenerator` class is an implementation of the abstract dungeon generator using a simple random walk algorithm. It extends the `AbstractDungeonGenerator` class and utilizes the `SimpleRandomWalkSO` (Scriptable Object) for configurable parameters.

4.1 `RunProceduralGeneration`

- **Description**: Executes the procedural generation process using the simple random walk algorithm.

4.2 `RunRandomWalk`

- **Description**: Executes the simple random walk algorithm with specified parameters.
- **Parameters**:
  - `parameters`: Scriptable Object containing parameters for the random walk.
  - `position`: Starting position for the random walk.
- **Returns:** A `HashSet<Vector2Int>` representing the floor positions generated by the random walk.

5. TreeDungeonGenerator

The `TreeDungeonGenerator` script is used for generating procedural dungeon layouts. It employs a tree-based structure to create interconnected rooms and corridors within specified areas. The generated dungeon includes various environmental elements such as platforms, enemies, spikes, chests, torches, trash bins, and more.

5.1 `Awake`

- **Description**: Initializes the singleton instance of the `TreeDungeonGenerator` and triggers procedural generation on Awake.

5.2 `RunProceduralGeneration`

- **Description**: This method serves as the entry point for the procedural generation of a game level. It involves several steps, including clearing existing data, setting the random seed, generating rooms and corridors in specific areas, connecting different areas with corridors, reorganizing corridors, generating environmental elements (such as enemies, platforms, spikes, etc.), and visualizing the level using a tile map:
  1. **Clear Existing Data:**
     - Clears various data structures, including `floor`, `wall`, `oneWayPlatforms`, `spaceInRoom`, `ladderPosiitonInRoom`, `spikes`, `tileMapVisualizer`, and `mapGameObjectPool`.
  2. **Set Random Seed:**
     - Sets the random seed for the procedural generation. If a specific seed is provided in the `DataHolder` instance, it uses that seed; otherwise, it generates a random seed.
  3. **Generate Rooms and Corridors:**
     - Calls `CreateRoomsInArea` for each area in the `areaList`, obtaining floor data, corridor data, room centers, and room lists for each area.
     - Combines the generated data into the global data structures (`floor`, `roomCenterList`, `corridors`, and `roomList`).
  4. **Connect Different Areas:**
     - Iterates over area pairs and creates corridors to connect them. Uses the `CreateCorridor` function to generate corridors between room centers in adjacent areas.

5. **Reorganize Corridors:**
    - Removes corridor tiles that overlap with floor tiles to ensure corridors are distinct.
6. **Generate Environments:**
    - Calls the `GenerateEnvironments` method to populate the level with various environmental elements such as enemies, platforms, spikes, etc.
7. **Create Floor:**
    - Uses the `tileMapVisualizer` to paint floor tiles based on the generated floor data.
8. **Create Ladders:**
    - Uses the `tileMapVisualizer` to paint ladder tiles on corridors.
9. **Create Platforms:**
    - Uses the `tileMapVisualizer` to paint platform tiles based on the generated one-way platforms.
10. **Create Spikes:**
    - Uses the `tileMapVisualizer` to paint spike tiles based on the generated spike data.
11. **Create Walls:**
    - Calls `WallGenerator.CreateWalls` to generate and visualize walls based on the floor data.

5.3 `GenerateEnvironments`

- **Description:** This function is responsible for generating environmental elements within each room. It iterates through the list of rooms (`roomList`) and performs the following tasks for each room:
    1. **Vertical Splitting:**
        - Calls the `SplitSpaceVertically` function to split the vertical space within the room and stores the resulting spaces in the `spaceInRoom` dictionary.
    2. **Door Generation:**
        - Checks if the room is in the last region (`areaList[areaList.Count - 1]`), and if so, generates a door using the `GenerateDoorInSpace` function.
    3. **Platform Generation:**
        - Determines whether a ladder is present in the room using the `ladderPositionInRoom` dictionary.
        - Calls the `GeneratePlatformInSpace` function to create platforms within the room based on certain conditions.
    4. **Birthplace Generation:**
        - Generates a birthplace in the first room if it has not been generated yet, using the `GenerateBirthPlace` function.
    5. **Enemy Generation:**
        - Calls the `GenerateEnemiesInSpace` function to populate the room with enemies, considering a threat level based on the enemy list.
    6. **Spike Generation:**
        - Calls the `GenerateSpikeInSpace` function to randomly generate spikes within the room based on a specified possibility.
    7. **Chest Generation:**

    - Calls the `GenerateChestsInSpace` function to randomly generate chests within the room based on a specified possibility.
8. **Torch Generation:**
    - Calls the `GenerateTorchesInSpace` function to generate torches within the room, with a maximum number defined by `torchNumInRoomMax`.
9. **Trash Bin Generation:**
    - Calls the `GenerateTrashBinInSpace` function to randomly generate trash bins within the room based on a specified possibility.

5.4 `SpliteSpaceHorizontallyWithSpecificArea`

- **Description**: Splits a given horizontal space with a specific area, resulting in two or more remaining spaces.
- **Parameters**:
    - `space`: The original space to be split horizontally (BoundsInt).
    - `area`: The specific area used for the horizontal split (BoundsInt).
- **Returns:** A list of BoundsInt representing the remaining spaces after the horizontal split.

5.5 `SpliteSpaceVertically`

- **Description**: Splits a given vertical space within a room, adding the resulting spaces to a HashSet.
- **Parameters**:
    - `room`: The original room space to be split vertically (BoundsInt).
    - `outputSpaceSet`: A HashSet to store the resulting spaces after the vertical split.

5.6 `GenerateBirthPlace`

- **Description**: This function attempts to generate a birthplace within a given room. It iterates through the spaces within the room, and if certain conditions are met, it creates a birthplace and updates the spatial information of the room.
- **Parameters**:
    - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
    - `room`: The room within which the birthplace is to be generated.
- **Returns:** A `bool` Indicates whether a birthplace was successfully generated.

5.7 `GenerateEnemysInSapce`

- **Description**: This function generates enemies within a given room based on the available spaces and a maximum threat value. It iterates through each space in the room, attempting to spawn enemies according to their properties and the remaining threat value.
- **Parameters**:
    - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
    - `room`: The room within which enemies are to be generated.
    - `threatenValueMax`: The maximum threat value that enemies collectively can have in the given room.

5.8 `GeneratePlatformInSpace`

- **Description**: This function generates platforms within a given room based on the available spaces. It iterates through each space in the room, and for each non-top space, it decides whether to generate a platform. If a platform is generated, it removes the topmost layer from the space and adds the platform to the list of one-way platforms. If a ladder position is provided, it also generates platforms around the ladder area.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which platforms are to be generated.
  - `possibility`: The likelihood of generating a platform within each eligible space.
  - `ladderPosition`: A tuple representing the ladder's horizontal position (optional).

5.9 `GenerateTorchsInSapce`

- **Description**: Generates torches within a specified room, considering the available spaces and a maximum number of torches.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which torches are to be generated.
  - `torchNumInRoomMax`: The maximum number of torches that can be generated in the given room.

5.10 `GenerateChestsInSapce`

- **Description**: Generates chests within a specified room, considering the available spaces and a given possibility.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which chests are to be generated.
  - `chestPossibility`: The probability of generating a chest in the given room.

5.11 `GenerateTrashBinInSpace`

- **Description**: Generates trash bins within a specified room, considering the available spaces and a given possibility.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which trash bins are to be generated.
  - `trashBinPossibility`: The probability of generating a trash bin in the given room.

5.12 `GenerateSpikeInSpace`

- **Description**: Generates spikes within a specified room, considering the available spaces and a given possibility.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which spikes are to be generated.
  - `possibility`: The probability of generating spikes in the given room.

5.13 `GenerateDoorInSapce`

- **Description**: Generates a door within a specified room, considering the available spaces.
- **Parameters**:
  - `spaceInRoom`: A dictionary containing spatial information for each room, including the available spaces.
  - `room`: The room within which a door is to be generated.

5.14 `CreateRoomsInArea`

- **Description**: Generates rooms and corridors within a specified area using binary space partitioning.
- **Parameters**:
  - `area` (BoundsInt): The area in which rooms and corridors will be generated.
- **Outputs:**
  - `floorInArea` (HashSet): Collection of floor tiles within the specified area.
  - `corridorsInArea` (HashSet[]): An array of two sets representing horizontal and vertical corridor tiles within the area.
  - `roomCenters` (List): List of centers of the generated rooms.
  - `roomListInArea` (List): List of BoundsInt representing the generated rooms within the area.

5.15 `CreateSimpleRooms`

- **Description**: Creates floor tiles within the generated rooms using a deterministic method.
- **Parameters**:
  - `roomList` (List): List of BoundsInt representing the generated rooms.
- **Outputs:**
  - `floor` (HashSet): Collection of floor tiles within the generated rooms.

5.16 `CreateCorridor`

- **Description**: Creates a corridor between two room centers.
- **Parameters**:
  - `currentCenter` (Vector2Int): The starting point of the corridor.
  - `destination` (Vector2Int): The destination point of the corridor.
  - `corridorWidthMin` (int): The minimum width of the corridor.
  - `corridorWidthMax` (int): The maximum width of the corridor.

    - `corridors` (HashSet[]): Array of two sets to store corridor tiles for each direction.
    - `ladderPosiitonInRoom` (Dictionary<Vector2Int, Tuple<int, int>>): Dictionary to store ladder positions in each room.
- **Outputs:**
    - Modifies the `corridors` and `ladderPosiitonInRoom` parameters to store corridor tiles and ladder positions.