Detailed Design

1. Introduction

This document describes the detailed design for the 'Modular World Generation' tool aimed at enabling Unity game developers to create scenes more efficiently and effectively. It details the components, architecture, and considerations necessary for the implementation of this tool.

2. System Overview

The system comprises two main components: the Algorithm Logic Side and the Database.

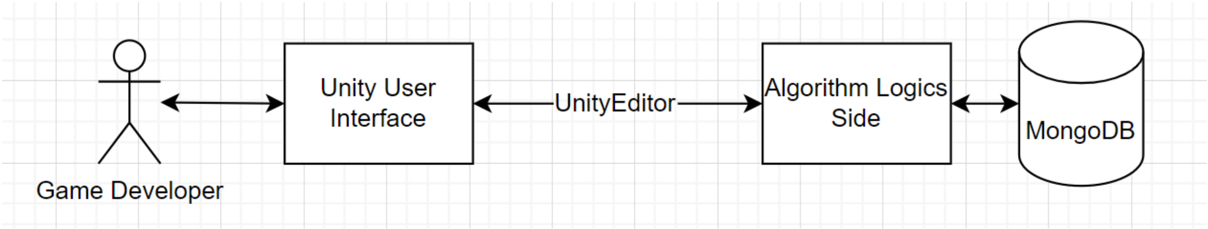
- **Algorithm Logic Side**: Responsible for receiving relevant data from the user to generate scenes, creating the scenes, and providing the final output.
- **Database**: Divided into two separate components: Assets and Scenes. The Assets component stores information about available assets to ensure organized work and efficiency. The 'Scene' component stores objects positioning for generation purposes.

The user interface is not a primary component of the system, as integration with the built-in Unity user interface is prioritized.

3. Design Considerations

- The system assumes the user is utilizing Unity engine version 5.x to ensure compatibility.
- The system assumes the user possesses the relevant assets.
- The system assumes all assets are anchor-centered for proper functionality.
- The system is designed exclusively for single-player game development (no multiplayer support).
- The algorithm logic side will primarily use C# and Unity libraries (e.g., UnityEditor).

4. System Architecture



5. Component Design

- **Algorithm Logic Side**:
    - **Purpose**: Provides generated scenes for the user.
    - **Input**: Preferences including the number and types of NPCs, buildings, and vehicles.
    - **Output**: Generated scene.
    - **Dependencies**: Requires connections with the Unity interface and the database.
- **Database**:
    - **Purpose**: Stores and provides information on assets and scenes.
    - **Input**: Relevant object (asset/scene).
    - **Output**: Information on relevant objects.
    - **Dependencies**: Requires connections with the Algorithm Logic Side.

6. Data Design - Database Schema

- **Collection: AssetInformation**
    - **Fields**:
        - **Asset_ID (int)**: Unique identifier for the asset. (Primary Key)
        - **Asset_name (string)**: Name of the asset. (Required)
        - **path_loc (string)**: Location path of the asset. (Required)
        - **Asset_width (double)**: Width of the asset. (Required)
        - **Asset_height (double)**: Height of the asset. (Required)
        - **Asset_length (double)**: Length of the asset. (Required)

- **Collection: Scene**
    - **Fields**:
        - **Location_X (double)**: X coordinate in space. (Composite Primary Key)
        - **Location_Y (double)**: Y coordinate in space. (Composite Primary Key)
        - **Location_Z (double)**: Z coordinate in space. (Composite Primary Key)
        - **Asset_ID (int)**: Foreign key referring to the asset. (Required)
        - **Offset_X (int)**: Offset value on the X axis. (Optional)
        - **Offset_Y (int)**: Offset value on the Y axis. (Optional)

** The primary key of the ‘Scene’ collection is a combination of Location_X, Location_Y, and Location_Z to maintain unique positioning in the scene space.

7. Detailed Class/Function Design

class NPC:

```
// Enum definitions for PersonalityType and SituationType
enum PersonalityType:
    GOOD
    BAD
    NEUTRAL

enum SituationType:
    NORMAL
    EMERGENCY
    PANIC
    HOSTILE
    FRIENDLY

// NPC Properties
interactable: bool
health: int
situationTag: SituationType
personalityTag: PersonalityType

// Physical Properties
size: Vector3[3]
scale: Vector3[3]
position: Vector3[3]
rotation: Vector3[3]
weight: float
walkingSpeed: float
runningSpeed: float

// Multimedia Components
animations: AnimationClip[]
audioSource: AudioSource

// Runtime variables
isInteracted: bool
rb: Rigidbody
navMeshAgent: NavMeshAgent
animator: Animator
destination: Vector3
isMoving: bool
isRunning: bool
```

```
// Awake function initializes the components
function Awake():
    rb = GetComponent(Rigidbody)
    navMeshAgent = GetComponent(NavMeshAgent)
    animator = GetComponent(Animator)

    // Initialize NavMeshAgent settings
    if navMeshAgent is not null:
        navMeshAgent.speed = walkingSpeed
        navMeshAgent.stoppingDistance = 1.0

// function that sets the ground for the ‘Immersive NPC Behavior System’ project
function UpdateSituation(newSituation: SituationType):
    situationTag = newSituation

    switch (newSituation):
        case SituationType.EMERGENCY:
            navMeshAgent.speed = runningSpeed
            isRunning = true
        case SituationType.PANIC:
            FindSafeSpot()
            isRunning = true
            navMeshAgent.speed = runningSpeed
        case SituationType.NORMAL:
            navMeshAgent.speed = walkingSpeed
            isRunning = false
        case SituationType.HOSTILE:
            PrepareForHostile()
        case SituationType.FRIENDLY:
            SetFriendlyBehavior()

    UpdateAnimationState()

function Interact(interactor: GameObject) -> bool:
    if not interactable or isInteracted:
        return false

    isInteracted = true
    HandleInteraction(interactor)
    return true

function SetDestination(newDestination: Vector3):
    if navMeshAgent is not null and navMeshAgent.isOnNavMesh:
        destination = newDestination
        navMeshAgent.SetDestination(destination)
        isMoving = true
        UpdateAnimationState()
```

```
function TakeDamage(damageAmount: int) -> bool:
    health -= damageAmount

    if health <= 0:
        Die()
        return false

    PlayHurtAnimation()
    return true

function Die():
    interactable = false
    if navMeshAgent is not null:
        navMeshAgent.enabled = false

    // Play death animation and sound
    PlayAnimation("Death")
    PlaySound("DeathSound")

// Handles different interactions based on personality and situation
function HandleInteraction(interactor: GameObject):
    switch (personalityTag):
        case PersonalityType.GOOD:
            HandleFriendlyInteraction(interactor)
        case PersonalityType.BAD:
            HandleHostileInteraction(interactor)
        case PersonalityType.NEUTRAL:
            HandleNeutralInteraction(interactor)



function UpdateAnimationState():
    if animator is not null:
        animator.SetBool("IsMoving", isMoving)
        animator.SetBool("IsRunning", isRunning)
        animator.SetInteger("SituationState", situationTag as int)

function PlayAnimation(animationName: string):
    if animator is not null:
        animator.Play(animationName)

function PlaySound(soundName: string):
    if audioSource is not null:
        audioSource.Play()

function FindSafeSpot():
    //gets from building class the positions of the safe spots of each building and return the
    nearest

// Draws debug visualization in the editor for developing purposes
function OnDrawGizmosSelected():
    Gizmos.color = Color.yellow
    Gizmos.DrawWireSphere(transform.position, 1.0)

    if navMeshAgent is not null and navMeshAgent.hasPath:
        Gizmos.color = Color.blue
        Gizmos.DrawLine(transform.position, destination)
```

```
class Tools:
  // Tool Properties
  grabable: bool
  movable: bool
  breakable: bool

  // Physical Properties
  size: Vector3[3]
  scale: Vector3[3]
  position: Vector3[3]
  rotation: Vector3[3]
  weight: float
  material: string

  // Multimedia Components
  animation: AnimationClip
  sound: AudioSource

  // Runtime variables
  isBroken: bool
  isGrabbed: bool
  rb: Rigidbody

  // Awake function initializes the Rigidbody component
  function Awake():
    rb = GetComponent(Rigidbody)

  function TryGrab() -> bool:
    if grabable and not isBroken and not isGrabbed:
      isGrabbed = true
      if rb is not null:
        rb.isKinematic = true
      return true
    return false

  function Release():
    if isGrabbed:
      isGrabbed = false
      if rb is not null:
        rb.isKinematic = false
```

```
function MoveTo(newPosition: Vector3, newRotation: Vector3) -> bool:
    if not movable or isBroken:
        return false

    transform.position = newPosition
    transform.eulerAngles = newRotation
    rotation = newRotation

    for i in range(0, position.length):
        position[i] = newPosition

    return true

function Break() -> bool:
    if breakable and not isBroken:
        isBroken = true
        PlayBreakEffects()
        return true
    return false


function PlayBreakEffects():
    if sound is not null:
        sound.Play()

    if animation is not null:
        // Get the Animation component and play the clip
        anim: Animation = GetComponent(Animation)
        if anim is not null:
            anim.AddClip(animation, "BreakAnimation")
            anim.Play("BreakAnimation")


// Gets the current status of the tool only for development and debugging
function GetStatus() -> string:
    return "Tool Status:\n" +
           "Material: " + material + "\n" +
           "Weight: " + weight + "\n" +
           "Is Broken: " + isBroken + "\n" +
           "Is Grabbed: " + isGrabbed + "\n" +
           "Can Move: " + (movable and not isBroken) + "\n" +
           "Position: " + transform.position + "\n" +
           "Rotation: " + rotation

// Gets the size measurements of the tool
function GetSizeMeasurements() -> Vector3[]:
    return size
```

class Building:

```
class Building:
   // Enum definitions for BuildingType
   enum BuildingType:
      RESTAURANT
      MALL
      STORE
      BANK

   // Building Properties
   type: BuildingType
   numberOfFloors: int
   capacity: int

   // Physical Properties
   size: Vector3[3]
   scale: Vector3[3]
   position: Vector3[3]
   rotation: Vector3[3]

   // Multimedia Components
   backgroundMusic: AudioSource
   ambientSounds: AudioSource[]

   // Dynamic Components
   buildingTools: List<Tools>
   occupants: List<NPC>

   // Runtime variables
   isOpen: bool
   currentOccupants: int
   activeAmbientSources: AudioSource[]

   function Awake():
      InitializeBuilding()

   function Start():
      SetupAudio()

   // Initializes the building's components and settings
   function InitializeBuilding():
      childTools: Tools[] = GetComponentsInChildren(Tools)
      childNPCs: NPC[] = GetComponentsInChildren(NPC)
      buildingTools.AddRange(childTools)
      occupants.AddRange(childNPCs)
      currentOccupants = occupants.Count
```

```
  // Sets up and initializes audio components
  function SetupAudio():
      if backgroundMusic is not null:
          backgroundMusic.loop = true
          backgroundMusic.playOnAwake = false

      activeAmbientSources = new AudioSource[ambientSounds.length]
      for i in range(0, ambientSounds.length):
          if ambientSounds[i] is not null:
              activeAmbientSources[i] = Instantiate(ambientSounds[i], transform.position,
              Quaternion.identity)
              activeAmbientSources[i].transform.parent = transform
              activeAmbientSources[i].loop = true
              activeAmbientSources[i].playOnAwake = false

  // Opens the building and starts its operations
  function OpenBuilding():
      isOpen = true
      PlayBackgroundMusic()
      PlayAmbientSounds()
      NotifyOccupants(true)

  // Closes the building and stops its operations
  function CloseBuilding():
      isOpen = false
      StopBackgroundMusic()
      StopAmbientSounds()
      NotifyOccupants(false)

  // Attempts to add an NPC to the building
  function AddOccupant(npc: NPC) -> bool:
      if not isOpen or currentOccupants >= capacity:
          return false

      occupants.Add(npc)
      currentOccupants++
      return true

  // Removes an NPC from the building
  function RemoveOccupant(npc: NPC):
      if occupants.Remove(npc):
          currentOccupants--

// Adds a tool to the building's inventory
  function AddTool(tool: Tools):
      if not buildingTools.Contains(tool):
          buildingTools.Add(tool)
          tool.transform.parent = transform
  // Removes a tool from the building's inventory
  function RemoveTool(tool: Tools):
      buildingTools.Remove(tool)

  // Gets the current occupancy status
  function GetOccupancyStatus() -> string:
      return "Current Occupants: " + currentOccupants + "/" + capacity +
             " (" + ((currentOccupants / capacity * 100).ToString("F1")) + "% full)"
```

```
class Vehicle:
// vehicle Properties
    parking: bool
    driving: bool

    // Physical Properties
    size: Vector3[3]
    scale: Vector3[3]
    position: Vector3[3]
    rotation: Vector3[3]

    // Multimedia Components
    animation: AnimationClip
    sound: AudioSource

    // Runtime variables
    isParking: bool
    isDriving: bool
    rb: Rigidbody
    destination: Vector3

    private Awake():
        rb = AddComponent<Rigidbody>()

    // Movement functions
    function StartDriving(targetDestination: Vector3):
        if not isDriving and not isParking:
            destination = targetDestination
            isDriving = true
            PlayAnimation("StartDriving")
            PlaySound("EngineStart")

    function StopDriving():
        if isDriving:
            isDriving = false
            PlayAnimation("StopDriving")
            PlaySound("EngineStop")

    function StartParking():
        if not isParking and not isDriving:
            isParking = true
            PlayAnimation("StartParking")
            PlaySound("ParkingBeep")
```

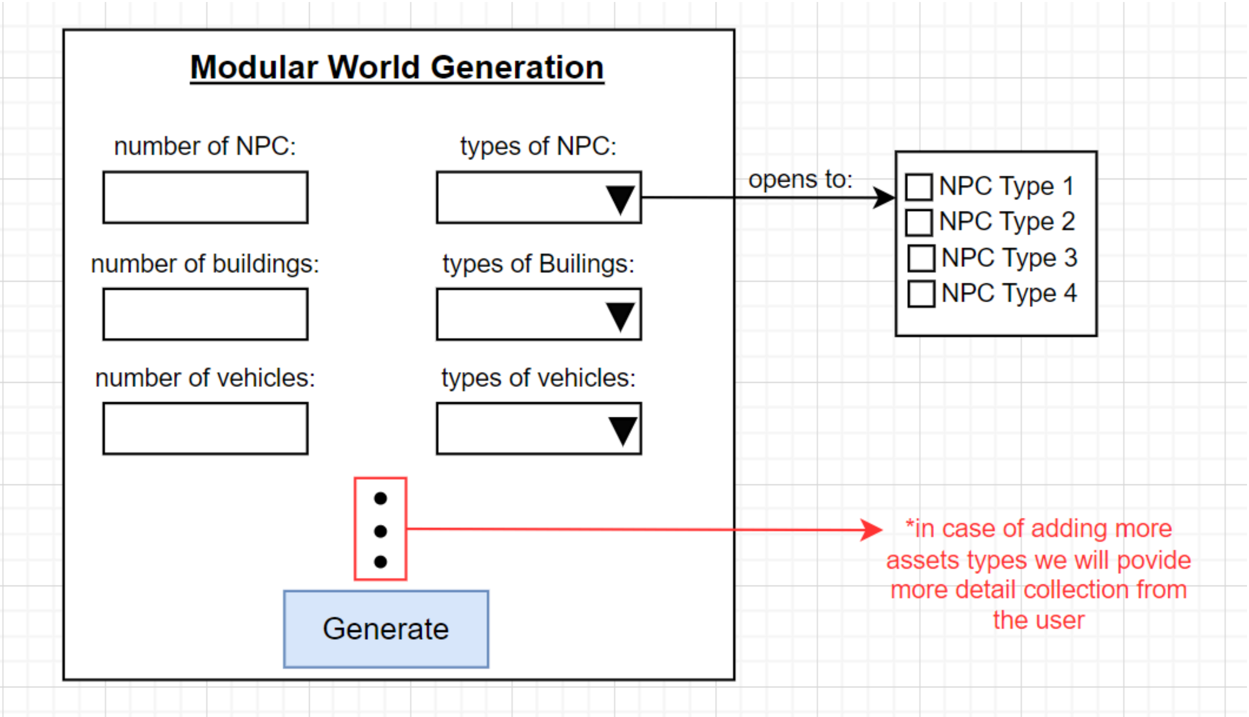
```
function FinishParking():
    if isParking:
        isParking = false
        parking = true
        PlayAnimation("ParkedIdle")
        PlaySound("ParkingComplete")

function UpdateVehicle():
    if isDriving:
        MoveTowardsDestination()
        UpdateRotation()
        CheckArrival()

function MoveTowardsDestination():
    direction = (destination - position).Normalize()
    velocity = direction * speedFactor
    position += velocity * deltaTime
    rb.MovePosition(position)

function UpdateRotation():
    targetRotation = CalculateTargetRotation()
    smoothRotation = LerpRotation(rotation, targetRotation, rotationSpeed * deltaTime)
    rotation = smoothRotation
    rb.MoveRotation(rotation)

function CheckArrival():
    if DistanceTo(destination) < arrivalThreshold:
        StopDriving()

// Animation and sound handlers
function PlayAnimation(animationName: string):
    if animation != null:
        animation.Play(animationName)

function PlaySound(soundName: string):
    if sound != null:
        sound.PlayOneShot(soundName)

function DistanceTo(target: Vector3) -> float:
    return Vector3.Distance(position, target)

function SetDestination(newDestination: Vector3):
    destination = newDestination
    if not isDriving:
        StartDriving(newDestination)
```

****NOTE:** From the versatile class, we can derive a variety of specialized sub-tools, such as chairs, tables, and plates for tools or civilian, officer, and thief for NPC ext. Each sub-class will have its own unique attributes and characteristics, tailored to fit its specific use and functionality.

```
interface IDynamicBehavior:

    // NPC-related functions
    UpdateSituation(newSituation: NPC.SituationType)
    Interact(interactor: GameObject) -> bool
    SetDestination(newDestination: Vector3)
    TakeDamage(damageAmount: int) -> bool
    Die()
    HandleFriendlyInteraction(interactor: GameObject)
    HandleHostileInteraction(interactor: GameObject)
    HandleNeutralInteraction(interactor: GameObject)
    FindSafeSpot()
    PrepareForHostile()
    SetFriendlyBehavior()
    PlayAnimation(animationName: string)
    PlaySound(soundName: string)

    // Vehicle-related functions
    StartDriving(targetDestination: Vector3)
    StopDriving()
    StartParking()
    FinishParking()
    UpdateVehicle()
    MoveTowardsDestination()
    UpdateRotation()
    CheckArrival()

    // Tool-related functions
    TryGrab() -> bool
    Release()
    MoveTo(newPosition: Vector3, newRotation: Vector3) -> bool
    Break() -> bool
    PlayBreakEffects()
```

8. User Interface Design

Since we are utilizing the built-in Unity user interface rather than creating a customized one, we are developing a window-type tool within Unity. Below is a mockup illustrating the intended design:


Modular World Generation
number of NPC:
types of NPC:
number of buildings:
types of Builings:
number of vehicles:
types of vehicles:
Generate
opens to:
□ NPC Type 1
□ NPC Type 2
□ NPC Type 3
□ NPC Type 4
*in case of adding more assets types we will povide more detail collection from the user