**KẾ HOẠCH PHÁT TRIỂN PROTOTYPE: ANIMAL RESCUE ADVENTURE**

**MỤC TIÊU PROTOTYPE**

1. **Xác nhận cơ chế gameplay chính** là thú vị và hấp dẫn
2. **Kiểm tra hiệu năng** trên các thiết bị di động mục tiêu
3. **Đánh giá độ khó** của các thử thách và câu đố theo nhóm tuổi
4. **Tạo nền tảng** để phát triển các tính năng và nội dung đầy đủ

**CẤU TRÚC DỰ ÁN UNITY**

Assets/

|-- \_Scenes/

| |-- MainMenu.unity

| |-- GameplayRunner.unity

| |-- PuzzleMode.unity

| |-- Sanctuary.unity

|

|-- Scripts/

| |-- Core/

| | |-- GameManager.cs

| | |-- DataManager.cs

| | |-- AudioManager.cs

| |

| |-- Player/

| | |-- PlayerController.cs

| | |-- PlayerCollision.cs

| | |-- PlayerAnimation.cs

| |

| |-- Level/

| | |-- LevelManager.cs

| | |-- ObstacleSpawner.cs

| | |-- EnvironmentManager.cs

| |

| |-- Puzzle/

| | |-- PuzzleManager.cs

| | |-- MinigameController.cs

| |

| |-- Sanctuary/

| | |-- SanctuaryManager.cs

| | |-- AnimalManager.cs

| |

| |-- UI/

| |-- UIManager.cs

| |-- MainMenuController.cs

| |-- HUDController.cs

|

|-- Prefabs/

| |-- Player/

| |-- Obstacles/

| |-- Environment/

| |-- Animals/

| |-- UI/

|

|-- Models/

|-- Animations/

|-- Materials/

|-- Textures/

|-- Audio/

|-- Resources/

**GIAI ĐOẠN PHÁT TRIỂN PROTOTYPE**

**1. Thiết lập Dự án (2-3 ngày)**

* **Tạo dự án Unity mới** (sử dụng Universal Render Pipeline)
* **Thiết lập Git repository** (với .gitignore phù hợp)
* **Cấu hình Unity cho phát triển mobile**:
  + Chọn Android & iOS là nền tảng mục tiêu
  + Tối ưu cài đặt Project Settings
  + Cấu hình Quality Settings ban đầu
* **Thiết lập cấu trúc thư mục** như đã nêu
* **Cài đặt các package cần thiết**:
  + Cinemachine (camera)
  + DoTween (animation)
  + TextMeshPro (UI text)
  + Device Simulator (test đa thiết bị)

**2. Runner Gameplay (7-10 ngày)**

**2.1 Thiết lập môi trường chạy**

// EnvironmentManager.cs (một phần)

public class EnvironmentManager : MonoBehaviour

{

[SerializeField] private GameObject[] groundPrefabs;

[SerializeField] private float segmentLength = 20f;

[SerializeField] private int segmentsToSpawn = 5;

[SerializeField] private int segmentsToKeep = 7;

private float playerMovementSpeed = 10f;

private List<GameObject> activeSegments = new List<GameObject>();

private Transform playerTransform;

private float spawnDistance;

private float despawnDistance;

private void Start()

{

playerTransform = GameObject.FindGameObjectWithTag("Player").transform;

spawnDistance = segmentLength \* (segmentsToSpawn - 1);

despawnDistance = -segmentLength \* 2;

// Khởi tạo phân đoạn môi trường đầu tiên

for (int i = 0; i < segmentsToSpawn; i++)

{

SpawnSegment();

}

}

private void Update()

{

// Kiểm tra khoảng cách để tạo thêm phân đoạn

if (playerTransform.position.z + spawnDistance >

activeSegments[activeSegments.Count - 1].transform.position.z)

{

SpawnSegment();

CheckForDespawn();

}

}

private void SpawnSegment()

{

Vector3 spawnPosition;

if (activeSegments.Count == 0)

{

spawnPosition = Vector3.zero;

}

else

{

spawnPosition = activeSegments[activeSegments.Count - 1].transform.position +

Vector3.forward \* segmentLength;

}

int randomIndex = Random.Range(0, groundPrefabs.Length);

GameObject segment = Instantiate(groundPrefabs[randomIndex],

spawnPosition, Quaternion.identity, transform);

activeSegments.Add(segment);

}

private void CheckForDespawn()

{

while (activeSegments.Count > segmentsToKeep)

{

GameObject segment = activeSegments[0];

activeSegments.RemoveAt(0);

Destroy(segment);

}

}

public float GetPlayerMovementSpeed()

{

return playerMovementSpeed;

}

public void SetPlayerMovementSpeed(float speed)

{

playerMovementSpeed = speed;

}

}

**2.2 Player Controller**

// PlayerController.cs (một phần)

public class PlayerController : MonoBehaviour

{

[Header("Movement")]

[SerializeField] private float jumpForce = 12f;

[SerializeField] private float slideSpeed = 8f;

[SerializeField] private float laneDistance = 3f; // Khoảng cách giữa các làn

[Header("Grounding")]

[SerializeField] private Transform groundCheckPoint;

[SerializeField] private float groundCheckRadius = 0.2f;

[SerializeField] private LayerMask groundLayer;

private Rigidbody rb;

private Animator animator;

private bool isJumping = false;

private bool isSliding = false;

private int currentLane = 1; // 0: left, 1: center, 2: right

private float targetLanePosition = 0;

private EnvironmentManager environmentManager;

private void Start()

{

rb = GetComponent<Rigidbody>();

animator = GetComponent<Animator>();

environmentManager = FindObjectOfType<EnvironmentManager>();

}

private void Update()

{

// Xử lý input

HandleInputs();

// Di chuyển nhân vật theo làn

HandleLaneMovement();

// Kiểm tra trạng thái trên mặt đất

CheckGrounded();

// Cập nhật animation

UpdateAnimations();

}

private void FixedUpdate()

{

// Di chuyển nhân vật về phía trước liên tục

rb.velocity = new Vector3(rb.velocity.x, rb.velocity.y,

environmentManager.GetPlayerMovementSpeed());

}

private void HandleInputs()

{

// Xử lý input để nhảy (vuốt lên)

if (SwipeDetector.Instance.SwipedUp && !isJumping && IsGrounded())

{

Jump();

}

// Xử lý input để trượt (vuốt xuống)

if (SwipeDetector.Instance.SwipedDown && !isSliding && IsGrounded())

{

StartCoroutine(Slide());

}

// Xử lý input để di chuyển sang trái (vuốt trái)

if (SwipeDetector.Instance.SwipedLeft && currentLane > 0)

{

currentLane--;

UpdateTargetLanePosition();

}

// Xử lý input để di chuyển sang phải (vuốt phải)

if (SwipeDetector.Instance.SwipedRight && currentLane < 2)

{

currentLane++;

UpdateTargetLanePosition();

}

}

private void Jump()

{

isJumping = true;

rb.velocity = new Vector3(rb.velocity.x, jumpForce, rb.velocity.z);

animator.SetTrigger("Jump");

}

private IEnumerator Slide()

{

isSliding = true;

// Thay đổi collider thành thấp hơn khi trượt

GetComponent<CapsuleCollider>().height = 0.5f;

GetComponent<CapsuleCollider>().center = new Vector3(0, 0.25f, 0);

animator.SetBool("Slide", true);

yield return new WaitForSeconds(1.0f); // Thời gian trượt

// Khôi phục collider

GetComponent<CapsuleCollider>().height = 1.8f;

GetComponent<CapsuleCollider>().center = new Vector3(0, 0.9f, 0);

animator.SetBool("Slide", false);

isSliding = false;

}

private void HandleLaneMovement()

{

// Di chuyển mềm mại giữa các làn

Vector3 targetPosition = new Vector3(targetLanePosition, transform.position.y, transform.position.z);

transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime \* 10f);

}

private void UpdateTargetLanePosition()

{

targetLanePosition = (currentLane - 1) \* laneDistance;

}

private bool IsGrounded()

{

return Physics.CheckSphere(groundCheckPoint.position, groundCheckRadius, groundLayer);

}

private void CheckGrounded()

{

if (IsGrounded() && isJumping && rb.velocity.y < 0.1f)

{

isJumping = false;

}

}

private void UpdateAnimations()

{

animator.SetBool("Grounded", IsGrounded());

animator.SetFloat("VerticalSpeed", rb.velocity.y);

}

}

**2.3 Chướng ngại vật và hệ thống sinh ngẫu nhiên**

// ObstacleSpawner.cs (một phần)

public class ObstacleSpawner : MonoBehaviour

{

[System.Serializable]

public class ObstacleData

{

public GameObject prefab;

public int difficulty; // 1-3 (dễ-khó)

public ObstacleType type;

}

public enum ObstacleType

{

Ground, // Chướng ngại vật trên mặt đất (cần trượt qua)

Air, // Chướng ngại vật trên không (cần nhảy qua)

Full, // Chướng ngại vật đầy làn (cần đổi làn)

Collectible // Vật phẩm thu thập

}

[SerializeField] private List<ObstacleData> availableObstacles = new List<ObstacleData>();

[SerializeField] private float minSpawnDistance = 5f; // Khoảng cách tối thiểu giữa các chướng ngại vật

[SerializeField] private float maxSpawnDistance = 15f; // Khoảng cách tối đa giữa các chướng ngại vật

[SerializeField] private float playerViewDistance = 100f; // Khoảng cách tối đa người chơi có thể nhìn thấy

private float nextSpawnDistance;

private float currentDifficulty = 1.0f; // Tăng theo thời gian

private Transform playerTransform;

private void Start()

{

playerTransform = GameObject.FindGameObjectWithTag("Player").transform;

CalculateNextSpawnDistance();

}

private void Update()

{

// Tăng độ khó theo thời gian chơi

currentDifficulty = Mathf.Min(3.0f, 1.0f + Time.timeSinceLevelLoad / 60.0f);

// Kiểm tra xem có cần sinh chướng ngại vật mới không

if (playerTransform.position.z + nextSpawnDistance < playerTransform.position.z + playerViewDistance)

{

SpawnObstacle();

CalculateNextSpawnDistance();

}

}

private void SpawnObstacle()

{

// Chọn chướng ngại vật dựa trên độ khó hiện tại

List<ObstacleData> eligibleObstacles = availableObstacles.Where(

o => o.difficulty <= Mathf.CeilToInt(currentDifficulty)).ToList();

if (eligibleObstacles.Count == 0)

return;

ObstacleData selectedObstacle = eligibleObstacles[Random.Range(0, eligibleObstacles.Count)];

// Xác định vị trí theo làn

int lane = Random.Range(0, 3); // 0: left, 1: center, 2: right

float lanePosition = (lane - 1) \* 3f; // 3f là khoảng cách giữa các làn

// Tính toán vị trí spawn

Vector3 spawnPosition = new Vector3(

lanePosition,

selectedObstacle.type == ObstacleType.Air ? 1.5f : 0f,

playerTransform.position.z + playerViewDistance

);

// Spawn chướng ngại vật

Instantiate(selectedObstacle.prefab, spawnPosition, Quaternion.identity, transform);

}

private void CalculateNextSpawnDistance()

{

// Khoảng cách giữa các chướng ngại vật giảm khi độ khó tăng

float difficultyFactor = 1.0f - (currentDifficulty - 1) / 4.0f;

float minAdjusted = minSpawnDistance \* difficultyFactor;

float maxAdjusted = maxSpawnDistance \* difficultyFactor;

nextSpawnDistance = Random.Range(minAdjusted, maxAdjusted);

}

}

**2.4 Hệ thống phát hiện vuốt (Swipe Detection)**

// SwipeDetector.cs

public class SwipeDetector : MonoBehaviour

{

public static SwipeDetector Instance;

[SerializeField] private float minSwipeDistance = 50f;

[SerializeField] private float maxSwipeTime = 0.5f;

private Vector2 startPos;

private float startTime;

private bool isSwiping = false;

// Properties để kiểm tra các hướng vuốt

public bool SwipedUp { get; private set; }

public bool SwipedDown { get; private set; }

public bool SwipedLeft { get; private set; }

public bool SwipedRight { get; private set; }

public bool Tapped { get; private set; }

private void Awake()

{

if (Instance == null)

{

Instance = this;

}

else

{

Destroy(gameObject);

}

}

private void Update()

{

// Reset các trạng thái vuốt mỗi frame

SwipedUp = SwipedDown = SwipedLeft = SwipedRight = Tapped = false;

#if UNITY\_EDITOR

// Xử lý input trong Unity Editor (sử dụng bàn phím)

HandleKeyboardInput();

#endif

// Xử lý input cảm ứng

HandleTouchInput();

}

private void HandleTouchInput()

{

if (Input.touchCount > 0)

{

Touch touch = Input.GetTouch(0);

switch (touch.phase)

{

case TouchPhase.Began:

startPos = touch.position;

startTime = Time.time;

isSwiping = true;

break;

case TouchPhase.Ended:

case TouchPhase.Canceled:

if (isSwiping)

{

float swipeTime = Time.time - startTime;

if (swipeTime < maxSwipeTime)

{

Vector2 endPos = touch.position;

Vector2 swipe = endPos - startPos;

float swipeLength = swipe.magnitude;

if (swipeLength > minSwipeDistance)

{

// Chuẩn hóa hướng vuốt

swipe.Normalize();

// Xác định hướng vuốt dựa trên góc

float angle = Mathf.Atan2(swipe.y, swipe.x) \* Mathf.Rad2Deg;

if (angle > -45 && angle <= 45)

{

SwipedRight = true;

}

else if (angle > 45 && angle <= 135)

{

SwipedUp = true;

}

else if (angle > 135 || angle <= -135)

{

SwipedLeft = true;

}

else

{

SwipedDown = true;

}

}

else

{

// Phát hiện tap (chạm nhanh)

Tapped = true;

}

}

}

isSwiping = false;

break;

}

}

}

#if UNITY\_EDITOR

private void HandleKeyboardInput()

{

// Sử dụng các phím mũi tên để mô phỏng vuốt khi phát triển trong Unity Editor

SwipedUp = Input.GetKeyDown(KeyCode.UpArrow);

SwipedDown = Input.GetKeyDown(KeyCode.DownArrow);

SwipedLeft = Input.GetKeyDown(KeyCode.LeftArrow);

SwipedRight = Input.GetKeyDown(KeyCode.RightArrow);

Tapped = Input.GetKeyDown(KeyCode.Space);

}

#endif

}

**3. System Mini-Game Puzzle (5-7 ngày)**

**3.1 Quản lý chuyển đổi giữa Runner và Puzzle**

// GameManager.cs (một phần)

public class GameManager : MonoBehaviour

{

public enum GameState

{

MainMenu,

Runner,

Puzzle,

Sanctuary,

Paused,

GameOver

}

private static GameManager \_instance;

public static GameManager Instance => \_instance;

[SerializeField] private GameObject runnerSceneObjects;

[SerializeField] private GameObject puzzleSceneObjects;

private GameState currentState;

private AnimalData currentAnimal;

private void Awake()

{

if (\_instance == null)

{

\_instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

}

}

private void Start()

{

ChangeState(GameState.MainMenu);

}

public void ChangeState(GameState newState)

{

currentState = newState;

switch (newState)

{

case GameState.MainMenu:

Time.timeScale = 1f;

// Hiển thị menu chính

break;

case GameState.Runner:

Time.timeScale = 1f;

EnableRunnerMode();

break;

case GameState.Puzzle:

Time.timeScale = 1f;

EnablePuzzleMode();

break;

case GameState.Sanctuary:

Time.timeScale = 1f;

// Chuyển đến scene khu bảo tồn

break;

case GameState.Paused:

Time.timeScale = 0f;

// Hiển thị menu tạm dừng

break;

case GameState.GameOver:

Time.timeScale = 0f;

// Hiển thị màn hình kết thúc

break;

}

UpdateUI();

}

private void EnableRunnerMode()

{

runnerSceneObjects.SetActive(true);

puzzleSceneObjects.SetActive(false);

}

private void EnablePuzzleMode()

{

runnerSceneObjects.SetActive(false);

puzzleSceneObjects.SetActive(true);

// Khởi tạo puzzle dựa trên animal hiện tại

if (currentAnimal != null)

{

FindObjectOfType<PuzzleManager>().InitializePuzzle(currentAnimal);

}

}

private void UpdateUI()

{

UIManager.Instance.UpdateUI(currentState);

}

// Được gọi khi gặp động vật cần cứu

public void TriggerAnimalRescue(AnimalData animal)

{

currentAnimal = animal;

ChangeState(GameState.Puzzle);

}

// Được gọi khi hoàn thành puzzle

public void CompletePuzzle(bool success)

{

if (success && currentAnimal != null)

{

// Thêm động vật vào khu bảo tồn

DataManager.Instance.AddRescuedAnimal(currentAnimal);

// Hiển thị thông báo thành công

UIManager.Instance.ShowRescueSuccess(currentAnimal);

}

// Quay lại chế độ Runner sau thời gian delay ngắn

Invoke("ReturnToRunner", 2.0f);

}

private void ReturnToRunner()

{

ChangeState(GameState.Runner);

}

}

**3.2 Quản lý puzzle và mini-game**

// PuzzleManager.cs (một phần)

public class PuzzleManager : MonoBehaviour

{

[System.Serializable]

public class PuzzleTemplate

{

public PuzzleType type;

public GameObject prefab;

public AgeGroup targetAge;

}

public enum PuzzleType

{

Matching,

Sorting,

Quiz,

Physics,

Memory

}

public enum AgeGroup

{

Young\_4\_7,

Mid\_8\_11,

Teen\_12\_15

}

[SerializeField] private List<PuzzleTemplate> puzzleTemplates = new List<PuzzleTemplate>();

[SerializeField] private Transform puzzleContainer;

private MinigameController activeMinigame;

private AgeGroup currentAgeGroup = AgeGroup.Mid\_8\_11; // Mặc định, sau này lấy từ cài đặt người chơi

public void InitializePuzzle(AnimalData animal)

{

// Xóa puzzle cũ nếu có

if (activeMinigame != null)

{

Destroy(activeMinigame.gameObject);

}

// Chọn puzzle phù hợp với độ tuổi

List<PuzzleTemplate> eligiblePuzzles = puzzleTemplates

.Where(p => p.targetAge == currentAgeGroup)

.ToList();

if (eligiblePuzzles.Count == 0)

return;

PuzzleTemplate selectedTemplate = eligiblePuzzles[Random.Range(0, eligiblePuzzles.Count)];

// Tạo instance của puzzle

GameObject puzzleInstance = Instantiate(selectedTemplate.prefab, puzzleContainer);

activeMinigame = puzzleInstance.GetComponent<MinigameController>();

// Khởi tạo puzzle với dữ liệu của động vật

activeMinigame.Initialize(animal);

}

// Được gọi khi hoàn thành puzzle

public void OnPuzzleCompleted(bool success)

{

GameManager.Instance.CompletePuzzle(success);

}

// Thiết lập nhóm tuổi

public void SetAgeGroup(AgeGroup age)

{

currentAgeGroup = age;

}

}

**3.3 Ví dụ về một mini-game đơn giản (Matching)**

// MatchingMinigame.cs

public class MatchingMinigame : MinigameController

{

[SerializeField] private GameObject cardPrefab;

[SerializeField] private Transform cardContainer;

[SerializeField] private int gridWidth = 4;

[SerializeField] private int gridHeight = 3;

private List<CardController> cards = new List<CardController>();

private CardController firstSelectedCard;

private CardController secondSelectedCard;

private int matchedPairs = 0;

private int totalPairs;

private AnimalData animalData;

private bool canSelect = true;

public override void Initialize(AnimalData animal)

{

animalData = animal;

CreateCards();

totalPairs = (gridWidth \* gridHeight) / 2;

matchedPairs = 0;

}

private void CreateCards()

{

// Xóa các thẻ cũ

foreach (Transform child in cardContainer)

{

Destroy(child.gameObject);

}

cards.Clear();

// Tạo danh sách các cặp hình ảnh

List<Sprite> cardImages = GetCardImages(gridWidth \* gridHeight / 2);

List<Sprite> cardPairs = new List<Sprite>();

// Mỗi hình ảnh sẽ xuất hiện hai lần

foreach (Sprite sprite in cardImages)

{

cardPairs.Add(sprite);

cardPairs.Add(sprite);

}

// Xáo trộn vị trí

ShuffleList(cardPairs);

// Khởi tạo grid thẻ

for (int y = 0; y < gridHeight; y++)

{

for (int x = 0; x < gridWidth; x++)

{

int index = y \* gridWidth + x;

if (index < cardPairs.Count)

{

GameObject cardObj = Instantiate(cardPrefab, cardContainer);

CardController card = cardObj.GetComponent<CardController>();

// Đặt vị trí

RectTransform rectTransform = cardObj.GetComponent<RectTransform>();

rectTransform.anchoredPosition = new Vector2(

x \* 120 - (gridWidth - 1) \* 60,

-y \* 120 + (gridHeight - 1) \* 60

);

// Thiết lập thẻ

card.Setup(cardPairs[index]);

card.OnCardClicked += OnCardSelected;

cards.Add(card);

}

}

}

}

private List<Sprite> GetCardImages(int count)

{

// Trong prototype, hãy sử dụng một số hình ảnh có sẵn

// Trong phiên bản đầy đủ, sẽ lấy hình ảnh liên quan đến động vật cụ thể

List<Sprite> availableImages = Resources.LoadAll<Sprite>("CardImages").ToList();

if (availableImages.Count < count)

{

Debug.LogWarning("Not enough card images available!");

count = Mathf.Min(count, availableImages.Count);

}

List<Sprite> selectedImages = new List<Sprite>();

for (int i = 0; i < count; i++)

{

selectedImages.Add(availableImages[i]);

}

return selectedImages;

}

private void ShuffleList<T>(List<T> list)

{

int n = list.Count;

for (int i = 0; i < n; i++)

{

int randomIndex = Random.Range(i, n);

T temp = list[i];

list[i] = list[randomIndex];

list[randomIndex] = temp;

}

}

private void OnCardSelected(CardController card)

{

if (!canSelect || card.IsMatched || card == firstSelectedCard)

return;

card.Flip();

if (firstSelectedCard == null)

{

firstSelectedCard = card;

}

else

{

secondSelectedCard = card;

canSelect = false;

// Kiểm tra xem hai thẻ có khớp không

CheckForMatch();

}

}

private void CheckForMatch()

{

bool isMatch = firstSelectedCard.GetImage() == secondSelectedCard.GetImage();

if (isMatch)

{

firstSelectedCard.SetMatched(true);

secondSelectedCard.SetMatched(true);

matchedPairs++;

// Kiểm tra chiến thắng

if (matchedPairs >= totalPairs)

{

Invoke("OnGameCompleted", 1.0f);

}

else

{

ResetSelection();

}

}

else

{

// Lật lại nếu không khớp

Invoke("FlipBackCards", 1.0f);

}

}

private void FlipBackCards()

{

firstSelectedCard.Flip();

secondSelectedCard.Flip();

ResetSelection();

}

private void ResetSelection()

{

firstSelectedCard = null;

secondSelectedCard = null;

canSelect = true;

}

private void OnGameCompleted()

{

PuzzleManager puzzleManager = FindObjectOfType<PuzzleManager>();

puzzleManager.OnPuzzleCompleted(true);

}

public override void OnTimeExpired()

{

PuzzleManager puzzleManager = FindObjectOfType<PuzzleManager>();

puzzleManager.OnPuzzleCompleted(false);

}

}

// CardController.cs

public class CardController : MonoBehaviour

{

[SerializeField] private Image frontImage;

[SerializeField] private Image backImage;

private bool isFlipped = false;

private bool isMatched = false;

private Sprite cardImage;

public event Action<CardController> OnCardClicked;

public void Setup(Sprite image)

{

cardImage = image;

frontImage.sprite = image;

isFlipped = false;

isMatched = false;

UpdateVisual();

}

public void Flip()

{

isFlipped = !isFlipped;

UpdateVisual();

}

public void SetMatched(bool matched)

{

isMatched = matched;

}

public bool IsMatched => isMatched;

public Sprite GetImage()

{

return cardImage;

}

private void UpdateVisual()

{

frontImage.gameObject.SetActive(isFlipped);

backImage.gameObject.SetActive(!isFlipped);

}

public void OnClick()

{

if (!isFlipped && !isMatched)

{

OnCardClicked?.Invoke(this);

}

}

}

**3.4 Interface chung cho mini-game**

// MinigameController.cs

public abstract class MinigameController : MonoBehaviour

{

[SerializeField] protected float timeLimit = 30f;

[SerializeField] protected Slider timerSlider;

protected float remainingTime;

protected bool isGameActive = false;

protected virtual void Start()

{

if (timerSlider != null)

{

timerSlider.maxValue = timeLimit;

timerSlider.value = timeLimit;

}

}

protected virtual void Update()

{

if (isGameActive)

{

UpdateTimer();

}

}

protected void UpdateTimer()

{

remainingTime -= Time.deltaTime;

if (timerSlider != null)

{

timerSlider.value = remainingTime;

}

if (remainingTime <= 0)

{

isGameActive = false;

OnTimeExpired();

}

}

public virtual void Initialize(AnimalData animal)

{

remainingTime = timeLimit;

isGameActive = true;

}

public abstract void OnTimeExpired();

}

**4. Hệ thống Dữ liệu và Quản lý (3-5 ngày)**

**4.1 AnimalData class**

// AnimalData.cs

[System.Serializable]

public class AnimalData

{

public string id;

public string animalName;

public string species;

public BiomeType biome;

public int rarity; // 1-5, 5 là hiếm nhất

public string description;

public string funFact;

public string iconPath;

public string modelPath;

public Dictionary<string, string> educationalContent = new Dictionary<string, string>();

public enum BiomeType

{

Jungle,

Ocean,

Savanna,

Arctic,

Mountain

}

// Constructor với các thông tin cơ bản

public AnimalData(string id, string name, string species, BiomeType biome)

{

this.id = id;

this.animalName = name;

this.species = species;

this.biome = biome;

this.rarity = 1;

}

// Thêm thông tin giáo dục theo độ tuổi

public void AddEducationalContent(string ageGroup, string content)

{

educationalContent[ageGroup] = content;

}

// Lấy thông tin giáo dục phù hợp với độ tuổi

public string GetEducationalContent(string ageGroup)

{

if (educationalContent.ContainsKey(ageGroup))

{

return educationalContent[ageGroup];

}

// Trả về thông tin mặc định nếu không có nội dung cho độ tuổi cụ thể

return description;

}

}

**4.2 DataManager class**

// DataManager.cs

public class DataManager : MonoBehaviour

{

private static DataManager \_instance;

public static DataManager Instance => \_instance;

// Danh sách tất cả động vật trong game

private List<AnimalData> allAnimals = new List<AnimalData>();

// Danh sách động vật đã được cứu

private List<AnimalData> rescuedAnimals = new List<AnimalData>();

// Dữ liệu người chơi

private PlayerData playerData = new PlayerData();

private void Awake()

{

if (\_instance == null)

{

\_instance = this;

DontDestroyOnLoad(gameObject);

// Khởi tạo dữ liệu

InitializeGameData();

LoadPlayerData();

}

else

{

Destroy(gameObject);

}

}

private void InitializeGameData()

{

// Trong prototype, khởi tạo một số động vật mẫu

// Trong phiên bản đầy đủ, sẽ tải từ JSON hoặc ScriptableObject

// Thêm động vật rừng nhiệt đới

AddSampleAnimal("monkey", "Khỉ", "Primate", AnimalData.BiomeType.Jungle);

AddSampleAnimal("parrot", "Vẹt", "Aves", AnimalData.BiomeType.Jungle);

AddSampleAnimal("jaguar", "Báo đốm", "Panthera onca", AnimalData.BiomeType.Jungle);

// Thêm động vật đại dương

AddSampleAnimal("dolphin", "Cá heo", "Delphinidae", AnimalData.BiomeType.Ocean);

AddSampleAnimal("turtle", "Rùa biển", "Cheloniidae", AnimalData.BiomeType.Ocean);

AddSampleAnimal("shark", "Cá mập", "Selachimorpha", AnimalData.BiomeType.Ocean);

}

private void AddSampleAnimal(string id, string name, string species, AnimalData.BiomeType biome)

{

AnimalData animal = new AnimalData(id, name, species, biome);

// Thêm thông tin giáo dục theo độ tuổi

animal.AddEducationalContent("4-7", "Thông tin đơn giản cho trẻ 4-7 tuổi về " + name);

animal.AddEducationalContent("8-11", "Thông tin chi tiết hơn cho trẻ 8-11 tuổi về " + name);

animal.AddEducationalContent("12-15", "Thông tin khoa học cho trẻ 12-15 tuổi về " + name);

animal.description = "Đây là mô tả về " + name;

animal.funFact = "Điều thú vị về " + name;

allAnimals.Add(animal);

}

private void LoadPlayerData()

{

// Trong prototype, chỉ khởi tạo dữ liệu mặc định

// Trong phiên bản đầy đủ, sẽ tải từ PlayerPrefs hoặc cloud save

playerData = new PlayerData();

// Trong prototype, đặt ageGroup mặc định

playerData.ageGroup = "8-11";

}

public void SavePlayerData()

{

// Trong phiên bản đầy đủ, sẽ lưu vào PlayerPrefs hoặc cloud save

Debug.Log("Saving player data...");

}

public void AddRescuedAnimal(AnimalData animal)

{

if (!rescuedAnimals.Contains(animal))

{

rescuedAnimals.Add(animal);

playerData.score += 100 \* animal.rarity; // Điểm thưởng dựa trên độ hiếm

SavePlayerData();

}

}

public List<AnimalData> GetAllAnimals()

{

return allAnimals;

}

public List<AnimalData> GetRescuedAnimals()

{

return rescuedAnimals;

}

public AnimalData GetAnimalById(string id)

{

return allAnimals.Find(a => a.id == id);

}

public AnimalData GetRandomAnimal(AnimalData.BiomeType biome)

{

List<AnimalData> biomeAnimals = allAnimals.FindAll(a => a.biome == biome);

if (biomeAnimals.Count > 0)

{

int randomIndex = Random.Range(0, biomeAnimals.Count);

return biomeAnimals[randomIndex];

}

return null;

}

public PlayerData GetPlayerData()

{

return playerData;

}

}

// PlayerData.cs

[System.Serializable]

public class PlayerData

{

public string playerName = "Ranger";

public int level = 1;

public int score = 0;

public string ageGroup = "8-11"; // 4-7, 8-11, 12-15

public int coins = 0;

public int stars = 0;

// Cài đặt

public float musicVolume = 1.0f;

public float sfxVolume = 1.0f;

public bool parentalControlsEnabled = false;

}

**5. UI cơ bản (3-5 ngày)**

**5.1 UIManager**

// UIManager.cs

public class UIManager : MonoBehaviour

{

private static UIManager \_instance;

public static UIManager Instance => \_instance;

[Header("UI Elements")]

[SerializeField] private GameObject mainMenuUI;

[SerializeField] private GameObject gameplayUI;

[SerializeField] private GameObject pauseMenuUI;

[SerializeField] private GameObject gameOverUI;

[SerializeField] private GameObject rescueSuccessUI;

[Header("HUD Elements")]

[SerializeField] private TextMeshProUGUI scoreText;

[SerializeField] private TextMeshProUGUI distanceText;

[SerializeField] private TextMeshProUGUI coinsText;

private void Awake()

{

if (\_instance == null)

{

\_instance = this;

}

else

{

Destroy(gameObject);

}

}

public void UpdateUI(GameManager.GameState state)

{

// Ẩn tất cả UI trước

mainMenuUI.SetActive(false);

gameplayUI.SetActive(false);

pauseMenuUI.SetActive(false);

gameOverUI.SetActive(false);

rescueSuccessUI.SetActive(false);

// Hiển thị UI tương ứng với trạng thái game

switch (state)

{

case GameManager.GameState.MainMenu:

mainMenuUI.SetActive(true);

break;

case GameManager.GameState.Runner:

case GameManager.GameState.Puzzle:

gameplayUI.SetActive(true);

break;

case GameManager.GameState.Paused:

gameplayUI.SetActive(true);

pauseMenuUI.SetActive(true);

break;

case GameManager.GameState.GameOver:

gameOverUI.SetActive(true);

break;

}

}

public void UpdateHUD(int score, float distance, int coins)

{

scoreText.text = score.ToString();

distanceText.text = Mathf.FloorToInt(distance).ToString() + "m";

coinsText.text = coins.ToString();

}

public void ShowRescueSuccess(AnimalData animal)

{

rescueSuccessUI.SetActive(true);

// Cập nhật thông tin động vật được cứu

TextMeshProUGUI animalNameText = rescueSuccessUI.transform.Find("AnimalName").GetComponent<TextMeshProUGUI>();

TextMeshProUGUI animalDescText = rescueSuccessUI.transform.Find("AnimalDescription").GetComponent<TextMeshProUGUI>();

Image animalImage = rescueSuccessUI.transform.Find("AnimalImage").GetComponent<Image>();

if (animalNameText != null)

animalNameText.text = animal.animalName;

if (animalDescText != null)

{

string ageGroup = DataManager.Instance.GetPlayerData().ageGroup;

animalDescText.text = animal.GetEducationalContent(ageGroup);

}

if (animalImage != null)

{

// Tải sprite của động vật nếu có

Sprite animalSprite = Resources.Load<Sprite>("Animals/" + animal.id);

if (animalSprite != null)

animalImage.sprite = animalSprite;

}

// Tự động ẩn sau 5 giây

Invoke("HideRescueSuccess", 5.0f);

}

private void HideRescueSuccess()

{

rescueSuccessUI.SetActive(false);

}

// Button event handlers

public void OnPlayButtonClicked()

{

GameManager.Instance.ChangeState(GameManager.GameState.Runner);

}

public void OnPauseButtonClicked()

{

GameManager.Instance.ChangeState(GameManager.GameState.Paused);

}

public void OnResumeButtonClicked()

{

GameManager.Instance.ChangeState(GameManager.GameState.Runner);

}

public void OnMainMenuButtonClicked()

{

GameManager.Instance.ChangeState(GameManager.GameState.MainMenu);

}

public void OnSanctuaryButtonClicked()

{

GameManager.Instance.ChangeState(GameManager.GameState.Sanctuary);

}

public void OnRestartButtonClicked()

{

// Reset game và bắt đầu lại

GameManager.Instance.ChangeState(GameManager.GameState.Runner);

}

public void OnQuitButtonClicked()

{

#if UNITY\_EDITOR

UnityEditor.EditorApplication.isPlaying = false;

#else

Application.Quit();

#endif

}

}

**6. Mobile Optimizations (2-3 ngày)**

**6.1 Tối ưu hiệu suất**

// PerformanceOptimizer.cs

public class PerformanceOptimizer : MonoBehaviour

{

[SerializeField] private int targetFrameRate = 60;

[SerializeField] private bool enableVSync = false;

private void Awake()

{

// Thiết lập target frame rate

Application.targetFrameRate = targetFrameRate;

// VSync (0 = không sử dụng, 1 = mỗi VBlank, 2 = mỗi 2 VBlank)

QualitySettings.vSyncCount = enableVSync ? 1 : 0;

// Tắt tự động sleep khi không focus

Application.runInBackground = true;

}

private void Start()

{

// Phát hiện và thiết lập quality settings phù hợp với thiết bị

DetectDeviceCapability();

}

private void DetectDeviceCapability()

{

// Phát hiện cấu hình thiết bị và thiết lập quality level phù hợp

// Đây là một cách tiếp cận đơn giản, trong phiên bản đầy đủ cần kiểm tra chi tiết hơn

int memorySize = SystemInfo.systemMemorySize;

int qualityLevel = QualitySettings.GetQualityLevel();

if (memorySize < 2048) // < 2GB RAM

{

qualityLevel = 0; // Low

}

else if (memorySize < 4096) // < 4GB RAM

{

qualityLevel = 1; // Medium

}

else

{

qualityLevel = 2; // High

}

QualitySettings.SetQualityLevel(qualityLevel, true);

// Log thông tin thiết bị để debugging

Debug.Log($"Device: {SystemInfo.deviceModel}, RAM: {memorySize}MB, GPU: {SystemInfo.graphicsDeviceName}");

Debug.Log($"Quality Level set to: {QualitySettings.names[qualityLevel]}");

}

public void AdjustQualityForBatteryLevel()

{

// Chỉ thực hiện trên thiết bị di động

#if UNITY\_ANDROID || UNITY\_IOS

// Kiểm tra mức pin

if (SystemInfo.batteryLevel < 0.2f) // < 20% pin

{

// Giảm frame rate để tiết kiệm pin

Application.targetFrameRate = 30;

// Giảm chất lượng đồ họa

int currentQuality = QualitySettings.GetQualityLevel();

if (currentQuality > 0)

{

QualitySettings.SetQualityLevel(currentQuality - 1, true);

}

Debug.Log("Battery low, reducing performance settings");

}

#endif

}

}

**6.2 Object Pooling**

// ObjectPool.cs

public class ObjectPool : MonoBehaviour

{

[System.Serializable]

public class Pool

{

public string tag;

public GameObject prefab;

public int size;

}

public static ObjectPool Instance;

public List<Pool> pools;

private Dictionary<string, Queue<GameObject>> poolDictionary;

private void Awake()

{

Instance = this;

poolDictionary = new Dictionary<string, Queue<GameObject>>();

foreach (Pool pool in pools)

{

Queue<GameObject> objectPool = new Queue<GameObject>();

for (int i = 0; i < pool.size; i++)

{

GameObject obj = Instantiate(pool.prefab);

obj.SetActive(false);

objectPool.Enqueue(obj);

obj.transform.SetParent(transform);

}

poolDictionary.Add(pool.tag, objectPool);

}

}

public GameObject SpawnFromPool(string tag, Vector3 position, Quaternion rotation)

{

if (!poolDictionary.ContainsKey(tag))

{

Debug.LogWarning("Pool with tag " + tag + " doesn't exist.");

return null;

}

GameObject objectToSpawn = poolDictionary[tag].Dequeue();

objectToSpawn.SetActive(true);

objectToSpawn.transform.position = position;

objectToSpawn.transform.rotation = rotation;

// Một số component có thể cần biết khi nào đối tượng được spawn lại

IPooledObject pooledObj = objectToSpawn.GetComponent<IPooledObject>();

if (pooledObj != null)

{

pooledObj.OnObjectSpawn();

}

poolDictionary[tag].Enqueue(objectToSpawn);

return objectToSpawn;

}

public void ReturnToPool(string tag, GameObject objectToReturn)

{

if (!poolDictionary.ContainsKey(tag))

{

Debug.LogWarning("Pool with tag " + tag + " doesn't exist.");

return;

}

objectToReturn.SetActive(false);

}

}

public interface IPooledObject

{

void OnObjectSpawn();

}

**7. Hệ thống âm thanh (2-3 ngày)**

// AudioManager.cs

public class AudioManager : MonoBehaviour

{

private static AudioManager \_instance;

public static AudioManager Instance => \_instance;

[System.Serializable]

public class Sound

{

public string name;

public AudioClip clip;

[Range(0f, 1f)]

public float volume = 1f;

[Range(0.1f, 3f)]

public float pitch = 1f;

public bool loop = false;

[HideInInspector]

public AudioSource source;

}

public Sound[] sounds;

private void Awake()

{

if (\_instance == null)

{

\_instance = this;

DontDestroyOnLoad(gameObject);

}

else

{

Destroy(gameObject);

return;

}

InitializeSounds();

}

private void InitializeSounds()

{

foreach (Sound s in sounds)

{

s.source = gameObject.AddComponent<AudioSource>();

s.source.clip = s.clip;

s.source.volume = s.volume;

s.source.pitch = s.pitch;

s.source.loop = s.loop;

}

}

public void Play(string name)

{

Sound s = Array.Find(sounds, sound => sound.name == name);

if (s == null)

{

Debug.LogWarning("Sound: " + name + " not found!");

return;

}

s.source.Play();

}

public void Stop(string name)

{

Sound s = Array.Find(sounds, sound => sound.name == name);

if (s == null)

{

Debug.LogWarning("Sound: " + name + " not found!");

return;

}

s.source.Stop();

}

public void UpdateVolume()

{

float musicVolume = DataManager.Instance.GetPlayerData().musicVolume;

float sfxVolume = DataManager.Instance.GetPlayerData().sfxVolume;

foreach (Sound s in sounds)

{

if (s.name.StartsWith("Music"))

{

s.source.volume = s.volume \* musicVolume;

}

else

{

s.source.volume = s.volume \* sfxVolume;

}

}

}

}

**KẾ HOẠCH KIỂM THỬ PROTOTYPE**

**1. Kiểm thử trên Unity Editor**

* Kiểm tra cơ chế chạy, nhảy, trượt
* Xác nhận sinh ngẫu nhiên chướng ngại vật hoạt động đúng
* Kiểm tra các mini-game giải câu đố
* Kiểm tra hiệu suất và tìm điểm nghẽn

**2. Kiểm thử trên thiết bị thực**

* Thử nghiệm trên ít nhất 2 thiết bị Android khác nhau (một tầm trung, một cao cấp)
* Kiểm tra điều khiển cảm ứng
* Xác nhận hiệu suất ổn định
* Kiểm tra tương thích với các kích thước màn hình

**3. Kiểm thử người dùng**

* Mời 3-5 người dùng trong nhóm tuổi mục tiêu thử nghiệm
* Thu thập phản hồi về:
  + Mức độ thú vị
  + Độ khó
  + Tính dễ sử dụng của giao diện
  + Khả năng hiểu được nội dung giáo dục

**DANH SÁCH CÔNG VIỆC**

**Tuần 1: Thiết lập & Runner Core**

1. Thiết lập dự án Unity và cấu hình cho mobile
2. Lập trình PlayerController cơ bản
3. Triển khai hệ thống tạo môi trường vô tận
4. Thêm hệ thống chướng ngại vật và va chạm
5. Tích hợp hệ thống điểm số cơ bản

**Tuần 2: Mini-game & Dữ liệu**

1. Thiết kế và triển khai chế độ puzzle đơn giản
2. Xây dựng hệ thống quản lý dữ liệu
3. Tạo mẫu data cho động vật
4. Thiết kế UI cơ bản
5. Tích hợp chuyển đổi giữa runner và puzzle

**Tuần 3: Tối ưu & Hoàn thiện**

1. Triển khai hệ thống âm thanh
2. Tối ưu performance cho thiết bị di động
3. Object pooling cho các gameobject thường xuyên được tạo/hủy
4. Sửa lỗi và cân bằng gameplay
5. Kiểm thử trên thiết bị thực và thu thập phản hồi

**KẾT QUẢ MONG ĐỢI**

Kết thúc giai đoạn prototype, chúng ta sẽ có:

1. **Runner gameplay** hoạt động mượt mà với:
   * Điều khiển nhảy, trượt, di chuyển sang làn
   * Hệ thống sinh ngẫu nhiên chướng ngại vật
   * Tương tác với động vật và vật phẩm
2. **Mini-game puzzle** đơn giản với:
   * 1-2 loại puzzle dễ triển khai (matching, sorting)
   * Hệ thống chuyển đổi giữa runner và puzzle
   * Thưởng khi hoàn thành thành công
3. **Dữ liệu mẫu**:
   * 5-6 loài động vật với thông tin cơ bản
   * Thông tin giáo dục theo độ tuổi
4. **UI cơ bản**:
   * Menu chính
   * HUD trong game
   * Màn hình kết quả
   * Thông báo cứu động vật thành công
5. **Tối ưu cho mobile**:
   * Chạy ổn định ở 30-60fps trên thiết bị tầm trung
   * Kích thước build < 100MB

Prototype này sẽ là nền tảng để chúng ta tiếp tục phát triển các tính năng đầy đủ hơn như khu bảo tồn, hệ thống tiến triển, và nội dung giáo dục phong phú hơn trong phiên bản tiếp theo.