Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Ingeniería de Software 1 (2016701)

Historia de Usuario #1

Descripción conceptual

Módulo	*Módulo de movimiento del personaje*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que el personaje se mueva de acuerdo a los inputs del usuario.*

Backend

Caso de uso técnico
Al recibir el input de teclado del usuario, se cambia el vector velocidad del personaje, moviéndolo.

Entrada
Input de teclado WASD
Implementación

```
    private void HandleMovement(float delta)
    {
        Vector2 direction = Vector2.Zero;
        if (Input.IsActionPressed("move_right"))
            direction.X += 1;
        if (Input.IsActionPressed("move_left"))
            direction.X -= 1;
        if (Input.IsActionPressed("move_down"))
            direction.Y += 1;
        if (Input.IsActionPressed("move_up"))
            direction.Y -= 1;
        direction = direction.Normalized();
        if (direction != Vector2.Zero)
        {
            _velocity = _velocity.Lerp(direction * Speed, Acceleration * delta);
        }
        else
        {
            _velocity = _velocity.Lerp(Vector2.Zero, Friction * delta);
        }
        Velocity = _velocity;
    }
}
```

Frontend

El personaje se mueve dependiendo de la dirección proporcionada, si el usuario presiona la tecla w, el personaje se mueve hacia arriba, si presiona la tecla a, se mueve hacia la izquierda, si presiona la tecla s, se mueve hacia abajo y si presiona la tecla d se mueve hacia la derecha.

Historia de Usuario #2

Descripción conceptual

Módulo	*Módulo de interacción con el entorno*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que el personaje sea capaz de interactuar con elementos en su entorno.*

Backend

Caso de uso técnico
Cuando el personaje se acerque a un objeto con el que pueda interactuar, se habilitará la interacción, además de mostrar un ícono que muestra esta posibilidad

Entrada
Input de teclado E
Implementación

```
public override void _Process(double delta)
{
    if (_playerNearby && Input.IsActionJustPressed("interact"))
    {
        Interact();
    }
}
private void Interact()
{
    GD.Print("Objeto interactuado");
        QueueFree(); }
```

Frontend

Cuando el personaje se acerca a un objeto con el que puede interactuar, el usuario presiona la tecla E en el teclado para interactuar con él. La interacción depende del objeto.

Historia de Usuario #3

Descripción conceptual

Módulo	*Módulo de interacción con el equipo*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que el personaje sea capaz de interactuar con los miembros de su equipo para asignar tareas*

Backend

Caso de uso técnico
Cuando se presione la tecla del equipo, se abre un menú donde se ve la información del equipo, los miembros y las tareas que se pueden asignar a ellos

Entrada
Input de teclado Q
Implementación

```
public partial class TeamManager : Node
{
    private List<TeamMember> _teamMembers = new List<TeamMember>();
    private Control _menuUI;
    private Label _teamInfoLabel;
    private bool _menuVisible = false;
    public override void _Ready()
    {
        _menuUI = GetNode<Control>("TeamMenu");
        _teamInfoLabel = GetNode<Label>("TeamMenu/TeamInfoLabel");
        _menuUI.Visible = false;
    }
    public override void _Process(double delta)
    {
        if (Input.IsActionJustPressed("open_team_menu"))
        {
            ToggleMenu();
        }
```

```
    }
    private void ToggleMenu()
    {
        _menuVisible = !_menuVisible;
        _menuUI.Visible = _menuVisible;
        if (_menuVisible)
        {
            UpdateTeamInfo();
        }
    }
    private void UpdateTeamInfo()
    {
        _teamInfoLabel.Text = "Miembros del equipo:\n";
        foreach (var member in _teamMembers)
        {
            _teamInfoLabel.Text += $"{member.Name} - {member.Role}\n";
            _teamInfoLabel.Text += "Tareas:\n";
            foreach (var task in member.Tasks)
            {
                _teamInfoLabel.Text += $"  - {task}\n";
            }
        }
    }
    public void AssignTask(string memberName, string task)
    {
        var member = _teamMembers.Find(m => m.Name == memberName);
        if (member != null)
        {
            member.AssignTask(task);
            UpdateTeamInfo();
        }
    }
}
```

Frontend

Cuando el usuario presiona la tecla Q, se muestra el menú con los miembros del equipo y las tareas que se les puede asignar. El menú se actualiza con la información dada, incluyendo nuevos miembros y nuevas tareas

Historia de Usuario #4

Descripción conceptual

Módulo	*Módulo de control de tareas*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que exista un menú donde estén todas las tareas, que se actualice dependiendo del estado de cada una.*

Backend

Caso de uso técnico

Al recibir el input de teclado J, se abre el menú de tareas, donde se muestran las tareas actuales y su estado.

Entrada

Input de teclado WASD

Implementación

```
public partial class TaskManager : Node
{
    private List<TaskData> _tasks = new List<TaskData>();
    private Control _taskMenu;
    private VBoxContainer _taskList;
    private bool _menuVisible = false;

    public override void _Ready()
    {
        _taskMenu = GetNode<Control>("TaskMenu");
        _taskList = GetNode<VBoxContainer>("TaskMenu/TaskList");
        _taskMenu.Visible = false;

    public override void _Process(double delta)
    {
        if (Input.IsActionJustPressed("open_task_menu"))
        {
            ToggleTaskMenu();
        }
    }

    private void ToggleTaskMenu()
    {
        _menuVisible = !_menuVisible;
        _taskMenu.Visible = _menuVisible;
        if (_menuVisible)
        {
            UpdateTaskList();
        }
    }

    private void UpdateTaskList()
```

```
    {
        foreach (Node child in _taskList.GetChildren())
        {
            child.QueueFree();
        }

        foreach (var task in _tasks)
        {
            var taskLabel = new Label();
            taskLabel.Text = $"{task.Title} - {task.Status}\n{task.Description}";
            _taskList.AddChild(taskLabel);
        }
    }

}
```

Frontend

Se abre el menú de tareas, donde el usuario puede revisar el estado de las tareas que tiene pendientes.

Historia de Usuario #5

Descripción conceptual

Módulo	*Módulo de control de recursos*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que se abra un menú donde el usuario pueda ver sus recursos y gestionarlos.*

Backend

Caso de uso técnico Al recibir el input de teclado I, se muestra una ventana con el inventario del personaje, desde la ventana se pueden gestionar los recursos que tiene el personaje.
Entrada Input de teclado WASD **Implementación** using Godot; using System.Collections.Generic; public partial class InventoryManager : Node

```
{
    private List<InventoryItem> _inventory = new List<InventoryItem>();
    private Control _inventoryMenu;
    private VBoxContainer _inventoryList;
    private bool _menuVisible = false;

    public override void _Ready()
    {
        _inventoryMenu = GetNode<Control>("InventoryMenu");
        _inventoryList = GetNode<VBoxContainer>("InventoryMenu/InventoryList");
        _inventoryMenu.Visible = false;
    }

    public override void _Process(double delta)
    {
        if (Input.IsActionJustPressed("open_inventory"))
        {
            ToggleInventoryMenu();
        }
    }
    private void ToggleInventoryMenu()
    {
        _menuVisible = !_menuVisible;
        _inventoryMenu.Visible = _menuVisible;
        if (_menuVisible)
        {
            UpdateInventoryList();
        }
    }
    private void UpdateInventoryList()
    {
        foreach (Node child in _inventoryList.GetChildren())
        {
            child.QueueFree();
        }

        foreach (var item in _inventory)
        {
            var itemLabel = new Label();
            itemLabel.Text = $"{item.Name} (x{item.Quantity}) - {item.Description}";
            _inventoryList.AddChild(itemLabel);
        }
    }

    public void UseItem(string itemName)
    {
        var item = _inventory.Find(i => i.Name == itemName);
        if (item != null && item.Quantity > 0)
        {
            item.UseItem();
            if (item.Quantity == 0)
            {
                _inventory.Remove(item);
            UpdateInventoryList();
```

```
    }
  }
}
```

Frontend

El usuario tiene la opción de abrir el menú de inventario, donde puede ver todos sus objetos además de poder usarlos o descartarlos.

Historia de Usuario #6

Descripción conceptual

Módulo	*Módulo de progreso*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que al final del sprint, se muestre una ventana enseñando el progreso del proyecto*

Backend

Caso de uso técnico
Cada 7 días dentro del juego, se muestra una ventana donde se enseña el progreso total del juego, además del estado de avance de los objetivos actuales.

Entrada
N/A
Implementación

```
public class ProgressManager : Node
{
  private const string SaveFilePath = "user://savegame.json";

  private class SaveData
  {
    public string LastProgressEvent { get; set; }
    public PlayerProgress PlayerProgress { get; set; }
  }

  private class PlayerProgress
  {
    public float TotalProgress { get; set; }
    public List<Objective> Objectives { get; set; }
  }
```

```
    private class Objective
    {
        public int Id { get; set; }
        public string Description { get; set; }
        public string Status { get; set; } // "completed", "inProgress", "failed"
    }

    private SaveData _currentSaveData;

    public override void _Ready()
    {
        LoadGameData();
        CheckProgressEvent();
    }


    private void LoadGameData()
    {
        if (File.Exists(SaveFilePath))
        {
            string jsonData = File.ReadAllText(SaveFilePath);
            _currentSaveData =
Newtonsoft.Json.JsonConvert.DeserializeObject<SaveData>(jsonData);
        }
        else
        {
            _currentSaveData = new SaveData
            {
                LastProgressEvent = DateTime.Now.ToString("o"),
                PlayerProgress = new PlayerProgress
                {
                    TotalProgress = 0,
                    Objectives = new List<Objective>
                    {
                        new Objective { Id = 1, Description = "Recolectar 100 monedas", Status =
"inProgress" },
                        new Objective { Id = 2, Description = "Derrotar 10 enemigos", Status =
"inProgress" }
                    }
                }
            };
            SaveGameData();
        }
    }

    private void SaveGameData()
    {
        string jsonData = Newtonsoft.Json.JsonConvert.SerializeObject(_currentSaveData);
        File.WriteAllText(SaveFilePath, jsonData);
    }

    private void CheckProgressEvent()
    {
```

```
        DateTime lastEvent = DateTime.Parse(_currentSaveData.LastProgressEvent);
        TimeSpan difference = DateTime.Now - lastEvent;

        if (difference.TotalDays >= 7)
        {
            ShowProgressWindow();
            _currentSaveData.LastProgressEvent = DateTime.Now.ToString("o");
            SaveGameData();
        }
    }

    private void ShowProgressWindow()
    {
        int totalObjectives = _currentSaveData.PlayerProgress.Objectives.Count;
        int completedObjectives = _currentSaveData.PlayerProgress.Objectives.FindAll(obj
=> obj.Status == "completed").Count;
        float progressPercentage = (completedObjectives / (float)totalObjectives) * 100;


        _currentSaveData.PlayerProgress.TotalProgress = progressPercentage;


        var progressWindow = GetNode<ProgressWindow>("ProgressWindow");
        if (progressWindow != null)
        {
            progressWindow.ShowProgress(progressPercentage,
_currentSaveData.PlayerProgress.Objectives);
        }
        else
        {
            GD.Print("Error: No se encontró la ventana de progreso.");
        }
    }

    public void UpdateObjectiveStatus(int objectiveId, string newStatus)
    {
        var objective = _currentSaveData.PlayerProgress.Objectives.Find(obj => obj.Id ==
objectiveId);
        if (objective != null)
        {
            objective.Status = newStatus;
            SaveGameData();
        }
    }
}
```

Frontend

Cada 7 días en el juego, se enseña la ventana de progreso, donde el usuario puede ver su progreso actual, además del estado de los objetivos que tiene pendientes.

Historia de Usuario #7

Descripción conceptual

Módulo	*Módulo de sugerencia de mejoras.*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que dentro de la pestaña de progreso, se enseñe una pestaña con sugerencias de mejoras en base a los objetos en el inventario y el progreso en el juego.*

Backend

Caso de uso técnico

Al encontrar la ventana de progreso, el juego dará una serie de sugerencias basadas en el progreso actual y el estado de los objetivos.

Entrada

N/A

Implementación

```
public List<string> GenerateSuggestions()
{
    List<string> suggestions = new List<string>();

    foreach (var objective in _currentSaveData.PlayerProgress.Objectives)
    {
        if (objective.Status == "inProgress")
        {
            switch (objective.Id)
            {
                case 1: // Ejemplo: Recolectar 100 monedas
                    suggestions.Add("Intenta explorar áreas nuevas para encontrar más
monedas.");
                    break;
                case 2: // Ejemplo: Derrotar 10 enemigos
                    suggestions.Add("Mejora tu equipo para derrotar enemigos más
fácilmente.");
                    break;
            }
        }
        else if (objective.Status == "failed")
        {
            suggestions.Add($"Revisa tu estrategia para el objetivo: {objective.Description}.");
        }
    }

    if (suggestions.Count == 0)
    {
        suggestions.Add("¡Sigue así! Estás haciendo un gran progreso.");
```

```
    }

    return suggestions;
}
```

Frontend

En el menú de progreso, el usuario tiene la oportunidad de ver un apartado donde se le indican sugerencias para mejorar sus resultados en el juego.

Historia de Usuario #8

Descripción conceptual

Módulo	*Módulo de gestión de recompensas*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que en la pestaña de objetivos exista un apartado donde se puedan reclamar las recompensas de los objetivos ya completados*

Backend

Caso de uso técnico
Cuando se abre la pestaña de los objetivos y encontrar objetivos completados, se abre la posibilidad de reclamar recompensas

Entrada
N/A
Implementación

```
public void CompleteTask(string taskTitle)
{
    var task = _tasks.Find(t => t.Title == taskTitle);
    if (task != null && task.Status == "inProgress")
    {
        task.Status = "completed";
        UpdateTaskList(); // Actualizar la lista de tareas
    }
}

private void OnClaimRewardPressed(TaskData task)
{
    if (task.Status == "completed")
    {
        _playerCoins += task.Reward;
```

```
            task.Status = "claimed";
            GD.Print($"¡Recompensa reclamada! Obtuviste {task.Reward} monedas. Total de
monedas: {_playerCoins}");
            UpdateTaskList();
        }
    }
```

Frontend

En la pestaña de tareas, si una tarea ya fue completada, el usuario tiene la oportunidad de reclamar las recompensas asociadas a dicha tarea.

Historia de Usuario #9

Descripción conceptual

Módulo	*Módulo de personalización del equipo*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que en la pantalla de inventario exista un apartado donde el jugador pueda ver y cambiar su equipo según su preferencia.*

Backend

Caso de uso técnico

Al enseñar la pantalla de inventario, hay un apartado con varias casillas de equipamiento, donde el usuario puede cambiar entre los equipos adquiridos hasta el momento.

Entrada

N/A

Implementación

```
public partial class EquipmentManager : Node
{
    public List<Item> Inventory { get; set; } = new List<Item>();
    public Equipment CurrentEquipment { get; set; } = new Equipment();

    // Método para añadir un ítem al inventario
    public void AddItem(Item item)
    {
        Inventory.Add(item);
        GD.Print($"Ítem añadido: {item.Name}");
    }

    // Método para equipar un ítem
    public void EquipItem(Item item)
```

```
    {
        switch (item.Type)
        {
            case "weapon":
                CurrentEquipment.Weapon = item;
                GD.Print($"Arma equipada: {item.Name}");
                break;
            case "armor":
                CurrentEquipment.Armor = item;
                GD.Print($"Armadura equipada: {item.Name}");
                break;
            case "accessory":
                CurrentEquipment.Accessory = item;
                GD.Print($"Accesorio equipado: {item.Name}");
                break;
            default:
                GD.Print("Tipo de ítem no válido.");
                break;
        }
    }
}
```

Frontend

En la interfaz de inventario, el usuario tiene la oportunidad de ver varias casillas de equipamiento, donde puede cambiar su equipo según sus necesidades.

Historia de Usuario #10

Descripción conceptual

Módulo	*Módulo de mejoras del reino*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que exista una interfaz donde el usuario pueda mejorar los diferentes aspectos del reino en función de cumplir los objetivos.*

Backend

Caso de uso técnico Al recibir el input de teclado T, se abre la interfaz de mejora del reino.
Entrada T **Implementación**

```
public partial class KingdomManager : Node
{
    public KingdomState KingdomState { get; set; } = new KingdomState();

    public KingdomManager()
    {
        KingdomState.Gold = 1000;
        KingdomState.Wood = 500;
        KingdomState.Stone = 300;

        KingdomState.Upgrades.Add(new KingdomUpgrade
        {
            Id = "upgrade_01",
            Name = "Mejora de Granja",
            Description = "Aumenta la producción de alimentos.",
            Cost = 200,
            IsUnlocked = false
        });

        KingdomState.Upgrades.Add(new KingdomUpgrade
        {
            Id = "upgrade_02",
            Name = "Mejora de Cantería",
            Description = "Aumenta la producción de piedra.",
            Cost = 300,
            IsUnlocked = false
        });
    }

    public bool PurchaseUpgrade(string upgradeId)
    {
        var upgrade = KingdomState.Upgrades.Find(u => u.Id == upgradeId);
        if (upgrade != null && !upgrade.IsUnlocked && KingdomState.Gold >= upgrade.Cost)
        {
            KingdomState.Gold -= upgrade.Cost;
            upgrade.IsUnlocked = true;
            GD.Print($"Mejora comprada: {upgrade.Name}");
            return true;
        }
        else
        {
            GD.Print("No se puede comprar la mejora. Verifica el coste o si ya está
desbloqueada.");
            return false;
        }
    }
}
```

Frontend

El usuario presiona la tecla T, entonces se abre la interfaz de mejora del reino, donde puede ver el estado de las partes del reino, las mejoras posibles y los recursos que podrían ser necesarios para la mejora, además de la posibilidad de mejorarlo.

Historia de Usuario #11

Descripción conceptual

Módulo	*Módulo de mejora de equipo.*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que el usuario pueda mejorar el equipo que tiene en una ubicación en específico.*

Backend

Caso de uso técnico
Cuando el usuario está en el herrero del reino, tiene la opción de mejorar su equipo a cambio de oro y materiales.

Entrada
N/A
Implementación

```
public partial class BlacksmithManager : Node
{
    public Equipment PlayerEquipment { get; set; } = new Equipment();
    public int Materials { get; set; } = 100; // Cantidad de materiales del jugador

    public List<BlacksmithUpgrade> AvailableUpgrades { get; set; } = new
List<BlacksmithUpgrade>
    {
        new BlacksmithUpgrade { ItemId = "sword_01", RequiredMaterials = 50, NewLevel =
2 },
        new BlacksmithUpgrade { ItemId = "armor_01", RequiredMaterials = 75, NewLevel =
2 }
    };

    public bool UpgradeItem(string itemId)
    {
        var upgrade = AvailableUpgrades.Find(u => u.ItemId == itemId);
        if (upgrade != null && Materials >= upgrade.RequiredMaterials)
        {
            Materials -= upgrade.RequiredMaterials;

            if (PlayerEquipment.Weapon?.Id == itemId)
            {
                PlayerEquipment.Weapon.Level = upgrade.NewLevel;
                GD.Print($"Arma mejorada a nivel {upgrade.NewLevel}.");
```

```
        }
        else if (PlayerEquipment.Armor?.Id == itemId)
        {
            PlayerEquipment.Armor.Level = upgrade.NewLevel;
            GD.Print($"Armadura mejorada a nivel {upgrade.NewLevel}.");
        }
        else if (PlayerEquipment.Accessory?.Id == itemId)
        {
            PlayerEquipment.Accessory.Level = upgrade.NewLevel;
            GD.Print($"Accesorio mejorado a nivel {upgrade.NewLevel}.");
        }

        return true;
    }
    else
    {
        GD.Print("No tienes suficientes materiales para mejorar este ítem.");
        return false;
    }
  }
}
```

Frontend

El usuario va a la ubicación del herrero, tiene la posibilidad de interactuar. Cuando interactúa con el herrero, se abre el menú de mejora del equipo, donde, a cambio de oro y materiales puede mejorar sus objetos.

Historia de Usuario #12

Descripción conceptual

Módulo	*Módulo de creación de objetos*
Descripción de la(s) funcionalidad(es) requerida(s):	*Se requiere que el usuario sea capaz de crear sus propios objetos y herramientas a cambio de materiales*

Backend

Caso de uso técnico
En la interfaz del herrero, se añade una alternativa para que el usuario cree sus propios objetos.
Entrada

N/A

Implementación

```
  public bool CraftItem(string itemId)
   {
      var recipe = CraftingRecipes.Find(r => r.ItemId == itemId);
      if (recipe != null && HasRequiredMaterials(recipe) && Gold >= recipe.CraftingCost)
      {

         foreach (var material in recipe.RequiredMaterials)
         {
            GD.Print($"Consumidos {material.Value} unidades de {material.Key}.");
         }
         Gold -= recipe.CraftingCost;

         var newItem = new Item
         {
            Id = recipe.ItemId,
            Name = recipe.ItemName,
            Type = "weapon", // O "armor", "accessory", etc.
            Level = 1
         };
         GD.Print($"Ítem creado: {newItem.Name}");

         return true;
      }
      else
      {
         GD.Print("No tienes los materiales o el oro necesario para crear este ítem.");
         return false;
      }
   }

   private bool HasRequiredMaterials(CraftingRecipe recipe)
   {
      foreach (var material in recipe.RequiredMaterials)
      {
          // Verificar si tiene los materiales
      }
      return true;
   }
}
```

Frontend

El usuario abre la interfaz del herrero, donde tiene la opción de crear sus propios objetos. Se le da la lista de recetas que tiene disponibles, el usuario tiene la opción de crear un objeto solo si tiene los materiales suficientes.