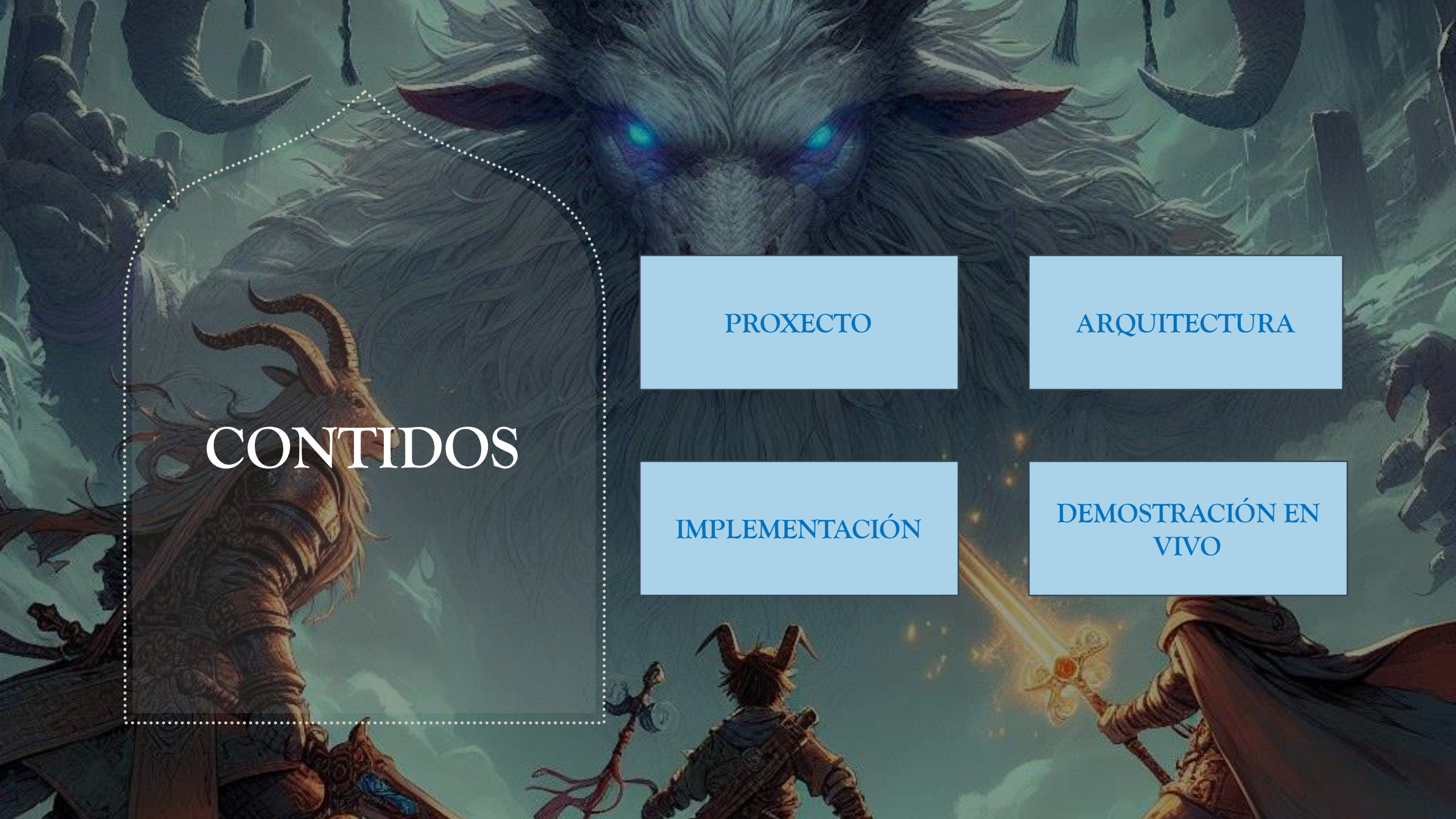
GOATS OF WAR

Pablo Fernández Pérez
José Manuel Amestoy López
Nicolás Rivas Rodríguez
Xoel González Pereira
Jordi Núñez Arias



CONTIDOS

- PROXECTO
- ARQUITECTURA
- IMPLEMENTACIÓN
- DEMOSTRACIÓN EN VIVO

PROXECTO

RPG Estratéxico

Combate por Turnos

Multixogador

Loitadores Intrépidos

Prepárate para o Combate!

REQUISITOS FUNCIONAIS E NON FUNCIONAIS

FUNCIONAIS

1. Rexistro de xogadores
2. Creación da partida
3. Combate por turnos
4. Interacción co xefe
5. Uso de pocións

NON FUNCIONAIS

1. Escalabilidade
2. Rendemento
3. Confiabilidade
4. Interconectividade
5. Tempo de resposta
6. Flexibilidade e mantenibilidade
7. Consistencia

ARQUITECTURA

DÚAS ARQUITECTURAS FUSIONADAS

LÍDER-TRABALLADOR

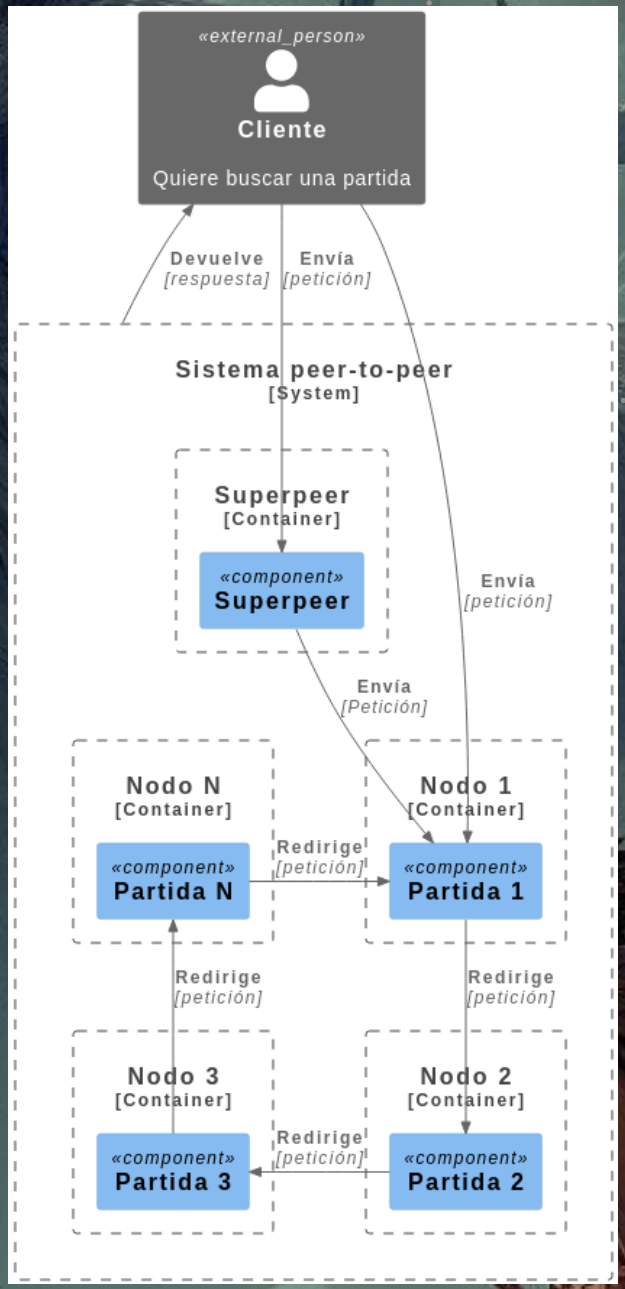
- A partida (que inclúe o xefe e toda a información) é o líder.
- O líder distribúe o traballo e ten lóxica de xestión.
- Os xogadores/personaxes son os traballadores.
- Os traballadores responden aos comandos do líder e executan accións específicas no xogo.

PEER-TO-PEER

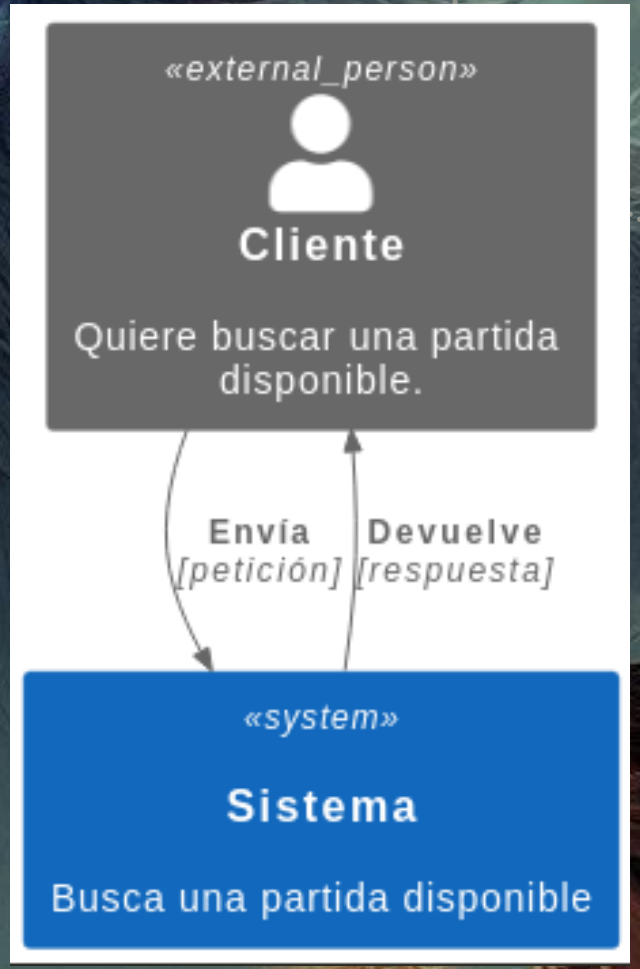
- Unha partida - un peer.
- Super-peer: crea as distintas partidas e busca unha dispoñible.
- Se unha partida non está dispoñible, podémonos unir a outra.

REPRESENTACIÓN C4

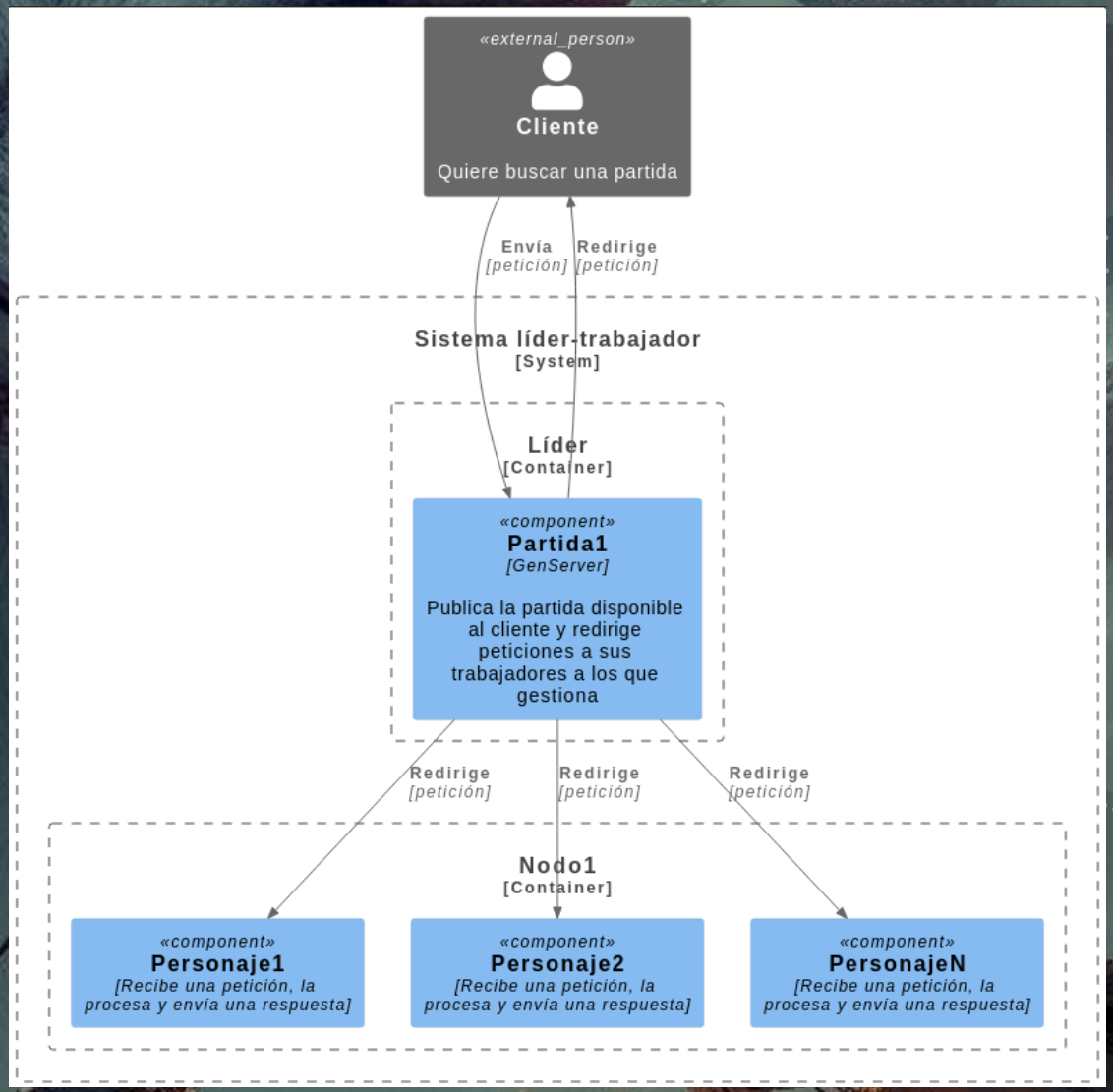
C4: Contedor


«external_person»
Cliente
Quiere buscar una partida
Devuelve
[respuesta]
Envía
[petición]
Sistema peer-to-peer
[System]
Superpeer
[Container]
«component»
Superpeer
Envía
[petición]
Envía
[Petición]
Nodo N
[Container]
«component»
Partida N
Redirige
[petición]
Nodo 1
[Container]
«component»
Partida 1
Redirige
[petición]
Redirige
[petición]
Nodo 3
[Container]
«component»
Partida 3
Redirige
[petición]
Nodo 2
[Container]
«component»
Partida 2

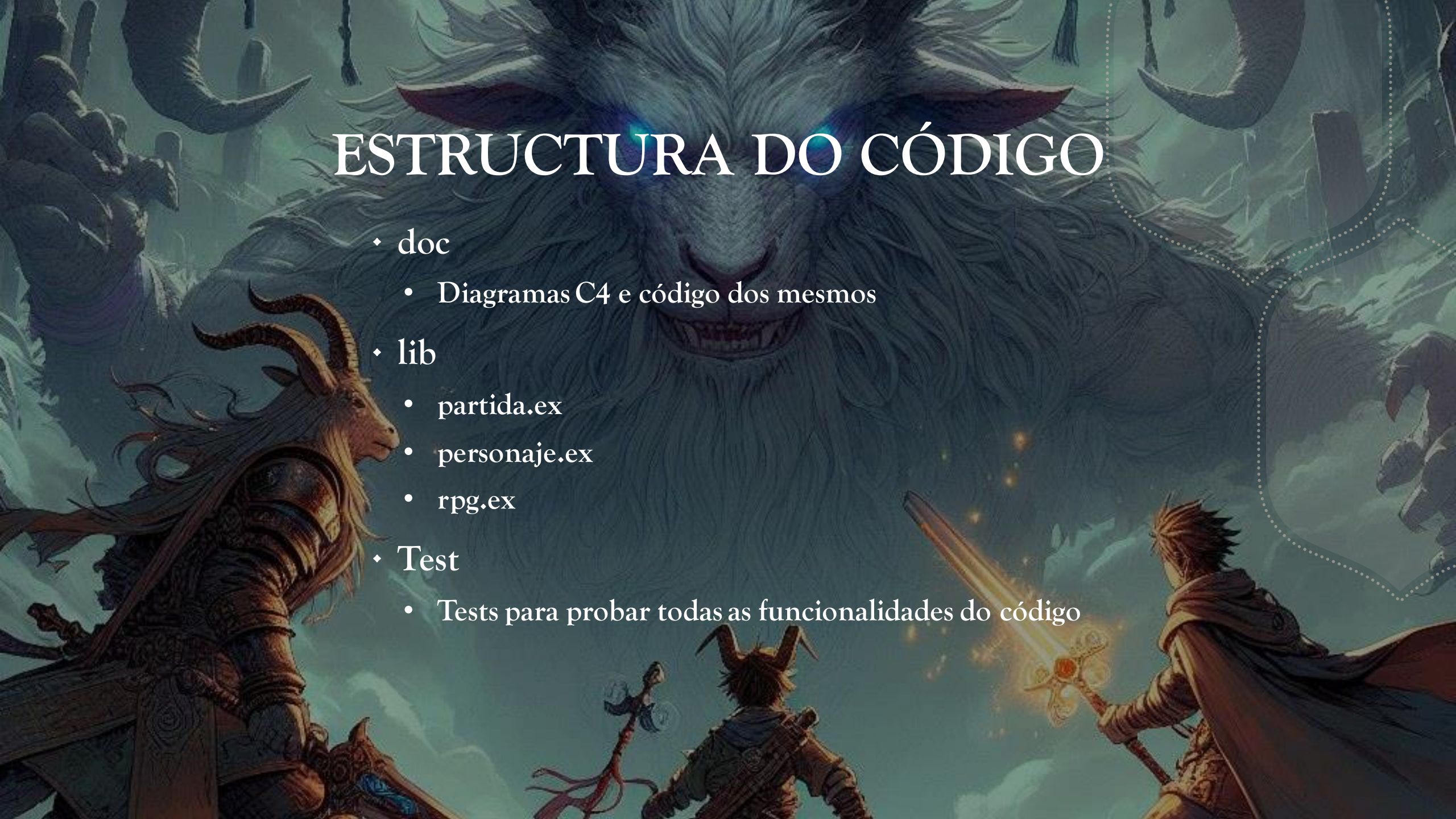
C4: Contexto


«external_person»
Cliente
Quiere buscar una partida disponible.
Envía
[petición]
Devuelve
[respuesta]
«system»
Sistema
Busca una partida disponible

C4: Compoñente



IMPLEMENTACIÓN



ESTRUCTURA DO CÓDIGO

- doc
  - Diagramas C4 e código dos mesmos
- lib
  - partida.ex
  - personaje.ex
  - rpg.ex
- Test
  - Tests para probar todas as funcionalidades do código

rpg.ex

Creación dos distintos peers

```elixir
def start_link(args) do
  GenServer.start_link(__MODULE__, args)
end

# Error si no se crea ningún peer
@impl true
def init(0) do
  {:stop, "Debe haber al menos un nodo."}
end

def init(numpeers) do
  {pids, _} = Enum.reduce_while(1..numpeers, {[], RPG.Partida}, fn _, {pids, partida_mod} ->
    {:ok, peer_pid} = RPG.Partida.start_link(@jugadores_max, 0, %{}, "AS", 2000, 2000, @tipos)
    Logger.info("Partida inicializada en peer #{inspect(peer_pid)} ")
    {:cont, {pids ++ [peer_pid], partida_mod}}
  end)

  {:ok, %{pids: pids}}
end
```

rpg.ex

Función para buscar unha partida dispoñible

```elixir
@impl true
def handle_call({:buscar_partida, tipo_personaje, nombre_personaje}, _from, state) do
  result =
    Enum.reduce_while(state.pids, {:ok, nil, nil}, fn pid, acc ->
      case GenServer.call(pid, {:unirse_a_partida, tipo_personaje, nombre_personaje}) do
        {:ok, j_pid, partida_pid} ->
          {:halt, {:ok, j_pid, partida_pid}}
          ->
          {:cont, acc}
      end
    end)

    case result do
      {:ok, j_pid, partida_pid} ->
        {:reply, {:ok, j_pid, partida_pid}, state}
        ->
        {:reply, {:error, "No hay partidas disponibles"}, state}
    end
end
end
```

partida.ex

Función para realizar un ataque especial

```elixir
def handle_cast({:ataque_especial, personaje_pid}, state) do
  case Map.get(state.jugadores, personaje_pid) do
    nil ->
      Logger.info("Error: No hay un jugador con ese nombre.")
      {:noreply, state}

    %{nombre: nombre_personaje, tipo: tipo_personaje} ->

      nuevo_estado_ataques = GenServer.call(personaje_pid, :obtener_estado)

      ataquesEspecialesRestantes = nuevo_estado_ataques.ataques_especiales

      if ataquesEspecialesRestantes != 0 do
        GenServer.cast(personaje_pid, {:ataquesEspecialesRestantes})

        daño_infligido = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 0)

        nueva_vida_jefe = state.vida_jefe - daño_infligido

        nuevo_estado = %{state | vida_jefe: nueva_vida_jefe}

        Logger.info("WoW, vaya ataque, #{nombre_personaje} ha infligido #{daño_infligido} de daño a #{state.nombre_jefe} (ง'̀-'́)ง")

        if nueva_vida_jefe <= 0 do
          Logger.info("¡Has derrotado a #{state.nombre_jefe}! ヾ(-_- )ゞ. Ahora te abrirá un hilo en Twitter. ᕦ(ò_óˇ)ᕤ")
          Logger.info("La partida ha terminado! Puedes volver a unirte si deseas volver a enfrentarte a #{state.nombre_jefe}!")
          nuevo_estado = %{state | vida_jefe: state.vida_inicial_jefe, jugadores: reiniciar_jugadores(state.jugadores)}
          {:noreply, nuevo_estado}
        else
          Logger.info("Vida actual de #{state.nombre_jefe}: #{nueva_vida_jefe}")
          GenServer.cast(personaje_pid, {:ataque_jefe, state.nombre_jefe})
          {:noreply, nuevo_estado}
        end
      else
        Logger.info(" Vaya no te quedan ataques especiales... ¯\\_(ツ)_/¯")
        {:noreply, state}
      end
  end
end
```

```elixir
def handle_cast({:atacar, personaje_pid}, state) do
  case Map.get(state.jugadores, personaje_pid) do
    nil ->
      Logger.info("Error: No hay un jugador con ese nombre.")
      {:noreply, state}

    %{nombre: nombre_personaje, tipo: tipo_personaje} ->
      daño_infligido = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 0)

      random_number = :rand.uniform(100)
      random_number2 = :rand.uniform(2)

      probabilidadAtaque = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 3)

      if random_number <= probabilidadAtaque do
        nueva_vida_jefe = state.vida_jefe - daño_infligido

        nuevo_estado = %{state | vida_jefe: nueva_vida_jefe}

        Logger.info(
          "#{nombre_personaje} ha infligido #{daño_infligido} de daño a #{state.nombre_jefe} (ง'̀-'́)ง"
        )

        if nueva_vida_jefe <= 0 do
          Logger.info("¡Has derrotado a #{state.nombre_jefe}! ヾ(-_- )ゞ. Ahora te abrirá un hilo en Twitter. ᕦ(ò_óˇ)ᕤ")
          Logger.info("La partida ha terminado! Puedes volver a unirte si deseas volver a enfrentarte a #{state.nombre_jefe}!")
          nuevo_estado = %{state | vida_jefe: state.vida_inicial_jefe, jugadores: reiniciar_jugadores(state.jugadores)}
          {:noreply, nuevo_estado}
        else
          Logger.info("Vida actual de #{state.nombre_jefe}: #{nueva_vida_jefe}")
          GenServer.cast(personaje_pid, {:ataque_jefe, state.nombre_jefe})
          {:noreply, nuevo_estado}
        end
      else
        if random_number2 == 1 do
          Logger.info("#{nombre_personaje} ha esquivado el ataque de #{state.nombre_jefe} (ಠ_ಠ)")
          GenServer.cast(personaje_pid, {:ataque_jefe, state.nombre_jefe})
          {:noreply, state}
        else
          Logger.info("#{nombre_personaje} falló el ataque a #{state.nombre_jefe}! (ノ°□°) ノ")
          GenServer.cast(personaje_pid, {:ataque_jefe, state.nombre_jefe})
          {:noreply, state}
        end
      end
  end
end
```

Función para realizar una ataque común

partida.ex

Función para unirse a unha partida

```elixir
def handle_call({:unirse_a_partida, tipo_personaje, nombre_personaje}, _from, state) do
  tipos_validos = Map.keys(state[:tipos])

  if state.numero_jugadores + 1 > state.capacidad_jugadores do
    Logger.info("Error: Se intentó unir a una partida llena. ¯\\_(⊙_⊙)_/¯ ")
    {:reply, {:error, "Partida llena"}, state}
  else
    if not Enum.member?(tipos_validos, tipo_personaje) do
      Logger.info("Error: Tipo de personaje no válido. Debes elegir entre espadachin, cabra o mago. (눈_눈)")
      {:reply, {:error, "Tipo de personaje no válido"}, state}
    else
      if Enum.any?(state.jugadores, fn {_, jugador} -> Map.get(jugador, :nombre) == nombre_personaje end) do
        Logger.info("Error: Ya hay un jugador con ese nombre. (⊙_⊙)")
        {:reply, {:error, "Nombre de jugador duplicado"}, state}
      else
        Logger.info("Uniéndose a partida #{inspect(self())} con #{state.numero_jugadores} jugador(es). Como #{nombre_personaje}, de tipo #{tipo_personaje} (ง'̀-'́)ง.")

        vida_inicial = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 1)
        num_pociones = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 2)
        ataques_especiales = elem(Map.get(state.tipos, tipo_personaje, {0, 0, 0, 0, 0}), 4)

        {:ok, personaje_pid} = RPG.Personaje.start_link(nombre_personaje, tipo_personaje, vida_inicial, vida_inicial, num_pociones, ataques_especiales, self())
        nuevo_estado = %{
          state |
          numero_jugadores: state.numero_jugadores + 1,
          jugadores: Map.put(state.jugadores, personaje_pid, %{tipo: tipo_personaje, vida: vida_inicial, nombre: nombre_personaje})
        }
        Logger.info("Lista de jugadores:")
        Enum.each(nuevo_estado.jugadores, fn {_pid, %{tipo: tipo, vida: vida, nombre: nombre_personaje}} ->
          Logger.info("- Nombre: #{nombre_personaje}, Tipo: #{tipo}, Vida: #{vida}")
        end)

        Logger.info("Vuestro objetivo es derrotar a #{state.nombre_jefe}")
        {:reply, {:ok, personaje_pid, self()}, nuevo_estado}
      end
    end
  end
end
```

Función para abandonar unha partida

```elixir
def handle_cast({:abandonar_partida, personaje_pid}, state) do
  case Map.get(state.jugadores, personaje_pid) do
    nil ->
      Logger.info("Error: Jugador no encontrado en la partida.")
      {:noreply, state}

    %{nombre: nombre_personaje} ->
      Logger.info("#{nombre_personaje} ha abandonado la partida. (╯︵╰,)")
      Process.exit(personaje_pid, :normal)

      nuevo_estado = %{state | jugadores: Map.delete(state.jugadores, personaje_pid)}
      {:noreply, nuevo_estado}
  end
end
```

personaje.ex

Función para atacar ao xefe

```
def handle_cast({:ataque_jefe, nombre_jefe}, state) do

    random_number = :rand.uniform(3)
    if random_number==3 do
      random_ataque = :rand.uniform(30)
      nuevo_estado = %{state | vida: max(state.vida - random_ataque, 0)}
      Logger.info("#{nombre_jefe} ha infligido #{random_ataque} de daño a #{state.nombre} ш (ﾟДﾟш) (╬ ಠ益ಠ)")
      if nuevo_estado.vida == 0 do
        Logger.info("#{state.nombre} ha muerto (✖╭╮✖) (DEP) (ಥ_ಥ)")
        GenServer.cast(state.partida_pid, {:abandonar_partida, self()})
        {:noreply, nuevo_estado}
      else
        Logger.info("A #{state.nombre} le quedan #{nuevo_estado.vida} puntos de vida ¯\\_(ツ)_/¯")
        {:noreply, nuevo_estado}
      end
    else if random_number==2 do
      Logger.info("#{state.nombre} ha esquivado el ataque de #{nombre_jefe} (ಠ益ಠ)")
      {:noreply, state}
    else
      Logger.info("#{nombre_jefe} falló el ataque a #{state.nombre}! (¯ﾍ¯)")
      {:noreply, state}
    end
  end
end
```

Función para usar unha poción

```
def handle_cast({:usar_pocion}, state) do
  if state.vida == state.vida_maxima do
    Logger.info("#{state.nombre}, tu vida ya está al máximo ( ˘ ³˘)♥")
    {:noreply, state}
  else
    if state.num_pociones == 0 do
      Logger.info("#{state.nombre}, no te quedan pociones, ¡Dios se apiade de ti! (´•⁔•`)")
      {:noreply, state}
    else
      nuevo_estado = %{state | vida: min(state.vida + 10, state.vida_maxima), num_pociones: state.num_pociones-1}
      Logger.info("#{state.nombre} ha usado una poción ( ˘ ³˘)♥. Nueva vida: #{nuevo_estado.vida}")
      if nuevo_estado.num_pociones == 0 do
        Logger.info("#{state.nombre}, no te quedan pociones, ¡Dios se apiade de ti! (´•⁔•`)")
      else
        Logger.info("#{state.nombre}, te quedan #{nuevo_estado.num_pociones} pocion(es). Adminístrala(s) bien! (u‿u)")
      end
      {:noreply, nuevo_estado}
    end
  end
end
```

DEMOSTRACIÓN


Agora é o momento de realizar a demostración
en vivo!