

Universidad Tecnológica de Honduras



Cátedra: Arquitectura de Computadoras

Catedrático: Ing. Ricardo Lagos

Juego Memory Game

Integrantes:

Oscar Alonso Reyes Álvarez **202410110040**

Jhonathan Josué Cruz **201910110505**

Patrik Mijaíl Mendoza Rodríguez **201730110180**

Kensy Rosio Rodríguez Ramos **202120120079**

Ingrys Dariela Santamaria Hernández **202110040149**

Fátima Isabel Briones Novoa **202110030119**

Denilson Josué Galo Osorto **201810110446**

Erikson Rodríguez Paz **202110010747**

Índice

Objetivos del Proyecto	3
Introducción	4
Análisis de las problemáticas	5
Justificación tecnológica	6
Pasos del Proyecto	6
Desarrollo de la Interfaz Gráfica con Python Django	7
Explicación Detallada de la Lógica del Juego	8
Diseño Técnico del Sistema	9
Documentación del Código	9
-Register.html: Es la página web para registrar nuevos jugadores	21
Permite a los usuarios crear una cuenta para acceder a todas las funciones del juego (jugar, ver perfil, etc.)	21
-Style.css: es un archivo de hojas de estilo en cascada (CSS) que se utiliza para definir la presentación y el diseño visual de las páginas web	29
Capturas del Funcionamiento del Juego	33
Conclusiones Técnicas	38

Objetivos del Proyecto

1. Garantizar la portabilidad y el aislamiento del entorno de desarrollo.
2. Asegurar la integridad y precisión de las estadísticas del jugador.
3. Seleccionar el framework web más adecuado para un desarrollo robusto y seguro.
4. Demostrar la aplicación práctica de conceptos de arquitectura de computadoras.

Introducción

Memory Game es una aplicación web interactiva desarrollada con el framework Django y desplegada mediante contenedores Docker. Este proyecto fue concebido con el objetivo de integrar conocimientos teóricos de la arquitectura de computadoras con una implementación práctica enfocada en la eficiencia, portabilidad y aislamiento del entorno.

Esta aplicación no es solo un juego de memoria: es una herramienta diseñada para aplicar conceptos clave como el procesamiento de datos, manejo de estados, lógica de control, y eficiencia computacional. Mediante su desarrollo, se abordan temas fundamentales de la arquitectura de computadores, permitiendo que el usuario experimente cómo se comporta la lógica a nivel práctico.

Análisis de las problemáticas

Durante la realización del proyecto se presentaron varios retos tanto técnicos como conceptuales. Las dificultades presentadas en el presente documento detallan a continuación;

- I. Entre las principales dificultades fue lograr la integración fluida entre la lógica del juego implementada en el backend (Django) y el comportamiento visual e interactivo en el frontend (HTML, CSS, JavaScript). Inicialmente, hubo dificultades para sincronizar los datos generados en Python con el tablero del juego renderizado en el navegador, especialmente para pasar de forma segura la información de las cartas al archivo game.js.
- II. Otro reto importante fue relacionado al uso de Docker. Aunque Docker proporciona un entorno aislado y reproducible, configurar correctamente los contenedores con Django, las dependencias de Python, los volúmenes y las variables de entorno requirió un esfuerzo significativo. Se detectaron errores por incompatibilidad de versiones y rutas incorrectas al mapear archivos estáticos.
- III. Además, durante la fase de persistencia de datos, se presentaron fallos al calcular correctamente las estadísticas del jugador, ya que al inicio no se filtraban bien las partidas completadas y se mezclaban con sesiones en curso. Esto provocaba resultados incorrectos en el perfil del jugador, problema que se resolvió implementando una validación más estricta en el modelo PlayerProfile.
- IV. Finalmente, se enfrentaron retos con el manejo del temporizador visual. En ciertos navegadores, el temporizador no se ejecutaba de forma sincronizada, lo que afectaba la experiencia de juego. Se solventó utilizando funciones nativas de JavaScript (setInterval) y una correcta lectura del atributo data-initial-time-limit para obtener valores precisos desde el HTML.

Estos inconvenientes permitieron reforzar el aprendizaje práctico de temas clave de arquitectura como aislamiento, manejo eficiente de recursos y sincronización entre capas lógicas del sistema.

Justificación tecnológica

Pasos del Proyecto

1. Configuración del entorno Docker: Se creó un Dockerfile y un docker-compose.yml para desplegar un entorno de desarrollo aislado.
2. Implementación del juego: Se creó una app Django llamada memory_game con modelos para usuarios y sesiones de juego.
3. Diseño de la interfaz: Se emplearon HTML, CSS, Bootstrap y JavaScript para una interfaz interactiva.
4. Integración de lógica de juego: Las reglas del juego se integraron en views.py, con control de intentos, tiempo y emparejamiento.
5. Estadísticas del usuario: Se calcularon victorias, derrotas, tiempo promedio y nivel más jugado mediante un modelo personalizado.

Desarrollo de la Interfaz Gráfica con Python Django

Se utilizaron plantillas HTML con soporte de Bootstrap para formularios de login, registro, selección de nivel, juego y perfil de usuario. El frontend incluye:

- **Temporizador visual:** La lógica del temporizador no se ejecuta en el servidor. En su lugar, el servidor Django pasa el límite de tiempo inicial (`game_time_limit`) al navegador cuando se carga la página del juego.

El código JavaScript del juego es el encargado de iniciar una cuenta regresiva en el navegador, actualizando el número de segundos en pantalla en tiempo real. Esto libera al servidor para que pueda atender a otros jugadores.

- **Sistema de intentos:** Al igual que el temporizador, el número de intentos inicial (`initial_attempts`) se define en el servidor de Django según el nivel de dificultad y se envía al navegador.

El control del contador de intentos se maneja por completo en el lado del cliente (en JavaScript). El contador se reduce cada vez que el jugador falla un par de cartas, y la interfaz se actualiza instantáneamente para mostrar el número restante.

- **Sonidos de victoria/derrota:** Los archivos de audio (música de fondo, efectos de victoria y derrota) se incluyen en la plantilla HTML que Django envía al navegador.

Cuando se produce un evento de victoria o derrota en el juego, el código JavaScript detecta ese evento y activa la reproducción del sonido correspondiente, ofreciendo una respuesta auditiva inmediata al jugador.

- **Interacción con cartas mediante JavaScript:** La manipulación de las cartas es el corazón de la interacción en el juego. El código JavaScript se encarga de todo, desde capturar los clics del usuario en las cartas, hasta animar el volteo de las mismas.

Este código también tiene la lógica para comparar las dos cartas seleccionadas, verificar si son un par, y gestionar la visibilidad de las cartas en el tablero (dejándolas boca arriba si coinciden o volteándolas si no lo hacen).

- **Reproducción de melodías y efectos sonoros:** Durante la experiencia del juego, incluyendo música de fondo mientras se juega, y sonidos distintivos para los eventos de victoria y derrota

Explicación Detallada de la Lógica del Juego

La lógica del juego se divide en tres partes, que trabajan juntas para ofrecer una experiencia completa.

Fase 1: Preparación (Servidor - views.py)

Cuando el jugador selecciona un nivel, la función `game_view` entra en acción. Esta función genera las cartas de forma aleatoria, asegurando que siempre haya pares y que el orden sea diferente en cada partida.

Simultáneamente, crea un registro de la partida en la base de datos (`GameSession`). Este registro es como el identificador único de la partida en curso, almacenando el nivel y la hora de inicio.

Fase 2: Ejecución (Cliente - game.js)

Toda la interacción del juego, como dar vuelta las cartas, verificar si son un par y gestionar el contador de intentos, sucede en el navegador del jugador.

El código JavaScript del juego decide cuándo el jugador ha ganado o perdido, basándose en si se encontraron todos los pares de cartas o si se agotaron los intentos y el tiempo.

Fase 3: Finalización (Servidor - game_end_api)

Una vez que el juego termina en el navegador, este envía una solicitud a la función `game_end_api` en el servidor.

Esta función recibe la información final (victoria o derrota, y la duración total) y actualiza la partida en la base de datos. Lo más importante es que también actualiza las estadísticas del jugador (`PlayerProfile`), incrementando las victorias o derrotas para llevar un registro preciso.

Diseño Técnico del Sistema

Documentación del Código

Cada archivo está documentado con comentarios para facilitar el mantenimiento:

-**Urls.py**: Es el sistema de enrutamiento de la aplicación. Se encarga de mapear las URL de tu sitio web a las funciones lógicas que se encuentran en el archivo

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('memory_game.urls')), # Incluye las URLs de tu app
]
```

Urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    # Autenticación y selección de nivel
    path('', views.login_view, name='login'), # Pantalla de login (ruta raíz)
    path('register/', views.register_view, name='register'), # Registro de usuario
    path('select_level/', views.select_level_view, name='select_level'), # Selección de nivel

    # Vistas principales del juego y perfil
    path('game/<str:level>', views.game_view, name='game'), # Vista del juego (inicia el juego para un nivel)
    path('profile/', views.profile_view, name='profile'), # Perfil del jugador
    path('logout/', views.logout_view, name='logout'), # Cerrar sesión

    path('game/move/', views.game_move_api, name='game_move_api'),
    # API para finalizar una partida y actualizar estadísticas
    path('game/end/<int:session_id>', views.game_end_api,
name='game_end_api'),

]
```

- **views.py**: Controla la autenticación, flujo del juego, actualización de datos.

```
# memory_game/views.py
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth import login, logout, authenticate
from django.contrib.auth.forms import UserCreationForm,
AuthenticationForm
from django.contrib.auth.decorators import login_required
from .models import PlayerProfile, GameSession # Asegúrate de que estos
modelos estén definidos en models.py
from django.http import JsonResponse, HttpResponseBadRequest # Importa
HttpResponseBadRequest para manejar errores de solicitud
import random
import json # Necesario para parsear el cuerpo JSON de las solicitudes
POST
from django.utils import timezone # Para obtener la hora actual con zona
horaria

# Vista de Login
def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                return redirect('select_level') # Redirige a selección de
nivel
        else:
            form = AuthenticationForm()
            return render(request, 'memory_game/login.html', {'form': form})

# Vista de Registro
def register_view(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            # Crea un perfil de jugador para el nuevo usuario
            #PlayerProfile.objects.create(user=user)
            login(request, user)
            return redirect('select_level')
        else:
```

```

        form = UserCreationForm()
        return render(request, 'memory_game/register.html', {'form': form})

# Vista de Selección de Nivel
@login_required
def select_level_view(request):
    return render(request, 'memory_game/select_level.html')

# Vista del Juego
@login_required
def game_view(request, level):
    # Lógica para inicializar el tablero, intentos, tiempo, etc.
    # Basado en el nivel seleccionado.
    num_cards = 0
    game_time_limit = 0
    initial_attempts = 0

    if level == 'Básico':
        num_cards = 8 # 4 pares de cartas
        game_time_limit = 60 # segundos
        initial_attempts = 10 # intentos
    elif level == 'Medio':
        num_cards = 12 # 6 pares de cartas
        game_time_limit = 90 # segundos
        initial_attempts = 8 # intentos
    elif level == 'Avanzado':
        num_cards = 16 # 8 pares de cartas
        game_time_limit = 120 # segundos
        initial_attempts = 6 # intentos
    else:
        # Redirige si el nivel no es válido
        return redirect('select_level')

    # Genera las cartas (pares de valores, por ejemplo, números o emojis)
    card_values = [str(i) for i in range(1, (num_cards // 2) + 1)] * 2
    random.shuffle(card_values)

    # Crea una nueva sesión de juego en la base de datos
    game_session = GameSession.objects.create(
        player=request.user,
        level=level,
        # start_time se establecerá automáticamente por auto_now_add en
        el modelo
    )

    context = {
        'level': level,
        'initial_attempts': initial_attempts,
        'cards': card_values,

```

```

        'game_session_id': game_session.id, # Pasa el ID de la sesión al
frontend
        'game_time_limit': game_time_limit, # Pasa el límite de tiempo al
frontend

    }
    return render(request, 'memory_game/game.html', context)

# Vista del Perfil
@login_required
def profile_view(request):
    profile, _ = PlayerProfile.objects.get_or_create(user=request.user)
    profile.update_statistics() # Fuerza actualización al entrar al
perfil
    game_history =
GameSession.objects.filter(player=request.user).order_by('-start_time')
    context = {
        'profile': profile,
        'game_history': game_history,
    }
    return render(request, 'memory_game/profile.html', context)

# Vista de Cerrar Sesión
@login_required
def logout_view(request):
    logout(request)
    return redirect('login')

# API para la lógica de un movimiento
def game_move_api(request):
    if request.method == 'POST' and request.user.is_authenticated:

        return JsonResponse({'status': 'success', 'message': 'Movimiento
procesado'})
        return JsonResponse({'status': 'error', 'message': 'Método no
permitido o no autenticado'}, status=400)

# API para finalizar una partida y actualizar estadísticas
def game_end_api(request, session_id):
    if request.method == 'POST':
        try:
            # 1. Parsea los datos JSON enviados desde el frontend
            data = json.loads(request.body)
            is_won = data.get('is_won')
            duration = data.get('duration')

            # 2. Obtiene la sesión de juego, verificando que pertenezca
al usuario actual

```

```

        game_session = get_object_or_404(GameSession, id=session_id,
player=request.user)

        # 3. Actualiza y guarda la sesión de juego
        game_session.is_won = is_won
        game_session.end_time = timezone.now()
        game_session.duration = duration
        game_session.save()

        # 4. Obtiene el perfil del jugador asociado al
usuario    total_games
        player_profile = get_object_or_404(PlayerProfile,
user=request.user)

        # 5. ACTUALIZA las estadísticas del perfil
        player_profile.games_played += 1
        if is_won:
            player_profile.total_wins += 1
        else:
            player_profile.total_losses += 1

        # 6. Guarda el perfil del jugador con las nuevas estadísticas
        player_profile.save()

        return JsonResponse({
            'status': 'success',
            'message': 'Sesión de juego finalizada y estadísticas
actualizadas',
            'is_won': game_session.is_won,
            'duration': game_session.duration
        })
    except json.JSONDecodeError:
        return HttpResponseBadRequest("Invalid JSON data in request
body.")
    except Exception as e:
        # Captura cualquier otro error, como si el perfil no existe
        return JsonResponse({'status': 'error', 'message': str(e)},
status=500)
    # Si el método no es POST
    return JsonResponse({'status': 'error', 'message': 'Método no
permitido'}, status=405)

```

- **models.py:** Define entidades GameSession y PlayerProfile.

```

# memory_game/models.py
from django.db import models
from django.contrib.auth.models import User

```

```

from django.db.models.signals import post_save
from django.dispatch import receiver
from collections import Counter
from django.db.models import Sum, Avg
import traceback

class PlayerProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    total_wins = models.IntegerField(default=0)
    total_losses = models.IntegerField(default=0)
    games_played = models.IntegerField(default=0)
    avg_time_per_game = models.FloatField(default=0.0)
    most_played_level = models.CharField(max_length=50, default='N/A')

    def __str__(self):
        return self.user.username

    def update_statistics(self):
        """
        Recalcula y actualiza todas las estadísticas del perfil del
jugador
basándose en las GameSessions asociadas.
        """
        try:
            # Filtra las sesiones de juego que están completas (is_won no
es null)
            # y tienen una duración registrada
            completed_sessions =
self.gamesession_set.filter(is_won__isnull=False, duration__isnull=False)

            self.games_played = completed_sessions.count()
            self.total_wins =
completed_sessions.filter(is_won=True).count()
            self.total_losses =
completed_sessions.filter(is_won=False).count()

            # Calcula el tiempo promedio usando la función de agregación
Avg
            avg_duration_agg =
completed_sessions.aggregate(Avg('duration'))['duration__avg']
            self.avg_time_per_game = avg_duration_agg if avg_duration_agg
is not None else 0.0

            # Determina el nivel más jugado
            levels = list(completed_sessions.values_list('level',
flat=True))
            if levels:
                from collections import Counter
                level_counts = Counter(levels)

```

```

        self.most_played_level =
level_counts.most_common(1)[0][0]
    else:
        self.most_played_level = 'N/A'

    # Guarda el perfil del jugador
    self.save()
    print(f"[DEBUG] Estadísticas actualizadas para
{self.user.username}")
except Exception as e:
    import traceback
    print(f"[ERROR] update_statistics: {e}")
    traceback.print_exc()

class GameSession(models.Model):
    player = models.ForeignKey(User, on_delete=models.CASCADE)
    level = models.CharField(max_length=50)
    start_time = models.DateTimeField(auto_now_add=True)
    end_time = models.DateTimeField(null=True, blank=True)
    duration = models.FloatField(null=True, blank=True)
    is_won = models.BooleanField(null=True, blank=True)
    attempts_left = models.IntegerField(null=True, blank=True)

    def __str__(self):
        status = 'Ganada' if self.is_won else 'Perdida' if self.is_won is
False else 'En Curso'
        return f"Sesión de {self.player.username} - {self.level}
({status})"

@receiver(post_save, sender=User)
def create_player_profile(sender, instance, created, **kwargs):
    if created:
        profile, created_profile =
PlayerProfile.objects.get_or_create(user=instance)
        print(f"[DEBUG] Perfil creado automáticamente para usuario:
{instance.username}, creado: {created_profile}")

@receiver(post_save, sender=GameSession)
def update_player_profile_on_game_session_save(sender, instance, created,
**kwargs):
    print(f"[DEBUG] Señal post_save GameSession para usuario:
{instance.player.username}, is_won: {instance.is_won}")
    # Solo actualiza si la partida ha finalizado (is_won no es null)
    if instance.is_won is not None and hasattr(instance.player,
'playerprofile'):
        print(f"[DEBUG] Llamando a update_statistics para
{instance.player.username}")

```

```

        instance.player.playerprofile.update_statistics()
    else:
        print(f"[DEBUG] No se actualizan estadísticas para
{instance.player.username}")

```

-Game.html: Es la plantilla HTML que crea la interfaz gráfica del juego. Este es el tablero visible. Muestra la información que le envía la función.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Memory Game - {{ level }}</title>

    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
    {% load static %}
    <link rel="stylesheet" href="{% static 'memory_game/css/style.css'
%}">

</head>
<body>
    <div class="container">
        <h1>Memory Game - Nivel {{ level }}</h1>
        <p>Intentos restantes: <span id="attempts-left">{{
initial_attempts }}</span></p>

        <p>Tiempo restante: <span id="time-left" data-initial-time-
limit="{{ game_time_limit }}">{{ game_time_limit }}</span> segundos</p>
        <div id="game-board" class="row">
            </div>
        </div>

        <audio id="victory-sound" src="{% static
'memory_game/sounds/victory.mp3' %}" preload="auto"></audio>
        <audio id="defeat-sound" src="{% static
'memory_game/sounds/defeat.mp3' %}" preload="auto"></audio>
        <audio id="background-music" src="{% static
'memory_game/sounds/background_music.mp3' %}" preload="auto"
loop></audio>
        {{ cards|json_script:"cards-data" }}

```



```

<script id="cards-data" type="application/json">

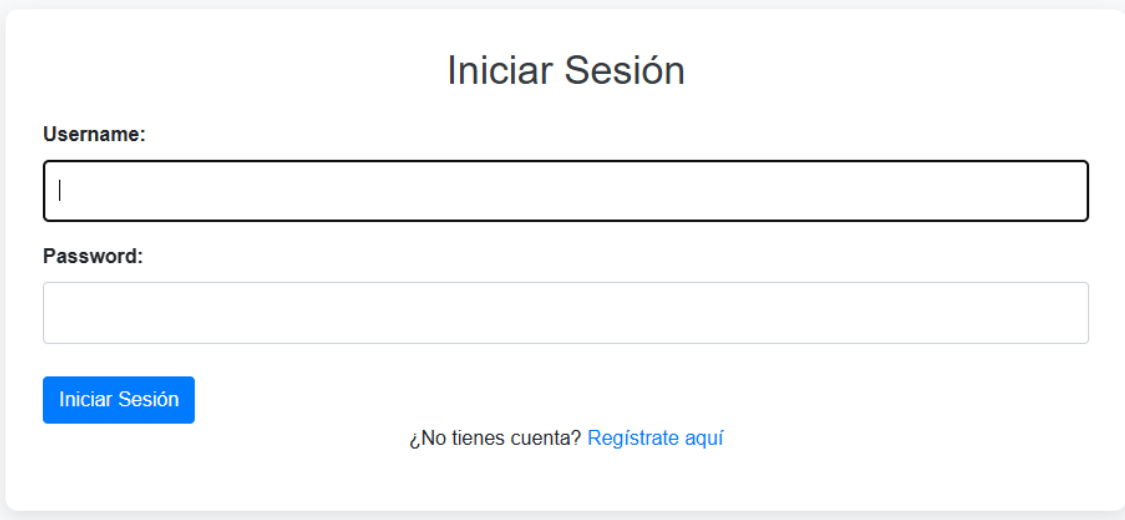
</script>
<script>
    const cardsData = JSON.parse(document.getElementById('cards-
data').textContent);
    const gameId = parseInt('{{ game_session_id }}');
    const csrfToken = "{{ csrf_token }}";
    const gameLevel = "{{ level }}"; // Pasar el nivel del juego
    const profileUrl = "{{ url 'profile' }}";
</script>

<script src="{{ static 'memory_game/js/game.js' }}"></script>

</body>
</html>

```

-Login.html: permite a los usuarios iniciar sesión.



Iniciar Sesión

Username:

Password:

[Iniciar Sesión](#)

¿No tienes cuenta? [Regístrate aquí](#)

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Login - Memory Game</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">

```

```
{% load static %}

<link rel="stylesheet" href="{% static 'memory_game/css/style.css'
%}">

</head>
<body>
    <div class="container">
        <h2>Iniciar Sesión</h2>
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <button type="submit" class="btn btn-primary">Iniciar
Sesión</button>
        </form>
        <p>¿No tienes cuenta? <a href="{% url 'register' %}">Regístrate
aquí</a></p>
    </div>
</body>
</html>
```

-Profile.html: muestra sus estadísticas y su historial de partidas guardado en la base de datos.

Perfil de lasbel

Estadísticas Generales:

Victorias Totales: 1

Derrotas Totales: 0

Partidas Jugadas: 1

Nivel más jugado: N/A

Historial de Partidas:

Fecha/Hora	Nivel	Duración (s)	Resultado
04 Aug 2025 02:26	Básico	13.00	<div>Victoria</div>

Volver a Selección de Nivel

Cerrar Sesión

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Perfil de Usuario - Juego de Memoria</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
  {% load static %}
```

```

</head>
<body>
  <div class="container mt-5">
    <div class="row justify-content-center">
      <div class="col-md-8">
        <div class="card">
          <div class="card-header text-center">
            <h2>Perfil de {{ user.username }}</h2>
          </div>
          <div class="card-body">
            <h4>Estadísticas Generales:</h4>
            <ul class="list-group mb-4">
              <li class="list-group-item">Victorias
Totales: {{ profile.total_wins }}</li>
              <li class="list-group-item">Derrotas Totales:
{{ profile.total_losses }}</li>
              <li class="list-group-item">Partidas Jugadas:
{{ profile.games_played }}</li>
              <li class="list-group-item">Tiempo promedio
por partida: {{ profile.avg_time_per_game|floatformat:2 }} segundos</li>
              <li class="list-group-item">Nivel más jugado:
{{ profile.most_played_level }}</li>
            </ul>

            <h4>Historial de Partidas:</h4>
            {% if game_history %}
            <div class="table-responsive">
              <table class="table table-striped table-
hover">
                <thead>
                  <tr>
                    <th>Fecha/Hora</th>
                    <th>Nivel</th>
                    <th>Duración (s)</th>
                    <th>Resultado</th>
                  </tr>
                </thead>
                <tbody>
                  {% for session in game_history %}
                  <tr>
                    <td>{{ session.start_time|date:"d
M Y H:i" }}</td>
                    <td>{{ session.level }}</td>
                    <td>{{
session.duration|floatformat:2|default:"N/A" }}</td>
                    <td>{% if session.is_won %}<span
class="badge badge-success">Victoria</span>{% else %}<span class="badge
badge-danger">Derrota</span>{% endif %}</td>

```

```

        </tr>
        {% endfor %}
    </tbody>
</table>
</div>
{% else %}
<p class="text-center">Aún no has jugado ninguna
partida.</p>

{% endif %}
<hr>
<div class="text-center">
    <a href="{% url 'select_level' %}" class="btn
btn-primary mt-3">Volver a Selección de Nivel</a>
    <a href="{% url 'logout' %}" class="btn btn-
secondary mt-3">Cerrar Sesión</a>
</div>
</div>
</div>
</div>
</div>
<script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.mi
n.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.
js"></script>
</body>
</html>

```

-Register.html: Es la página web para registrar nuevos jugadores.

Permite a los usuarios crear una cuenta para acceder a todas las funciones del juego (jugar, ver perfil, etc.).

Usa un formulario para capturar los datos del nuevo jugador y los envía al servidor para ser procesados y guardados en la base de datos del juego.

Después de un registro exitoso, redirige al jugador a la pantalla de selección de nivel para empezar a jugar.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Registrarse - Juego de Memoria</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
  {% load static %}

</head>
<body>
  <div class="container mt-5">
    <div class="row justify-content-center">
      <div class="col-md-6">
        <div class="card">
          <div class="card-header text-center">
            <h2>Registrarse</h2>
          </div>
          <div class="card-body">
            <form method="post">
              {% csrf_token %}
              {{ form.as_p }}
              <button type="submit" class="btn btn-success
btn-block">Registrarse</button>
            </form>
            <hr>
            <p class="text-center">¿Ya tienes una cuenta? <a
href="{% url 'login' %}">Inicia sesión aquí</a></p>
          </div>
        </div>
      </div>
    </div>
  </div>
  <script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
  <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.mi
n.js"></script>
  <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.
js"></script>
</body>
</html>

```

-Select_level.html: les permite elegir la dificultad

Hola, lasbel!

Selecciona un Nivel de Dificultad

Nivel Básico (10 intentos, 8 cartas, 60s)

Nivel Medio (8 intentos, 12 cartas, 90s)

Nivel Avanzado (6 intentos, 16 cartas, 120s)

Ver mi Perfil

Cerrar Sesión

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Seleccionar Nivel - Juego de Memoria</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.mi
n.css">
  {% load static %}
</head>
<body>
  <div class="container mt-5">
    <div class="row justify-content-center">
      <div class="col-md-8">
        <div class="card">
          <div class="card-header text-center">
            <h2>Hola, {{ user.username }}!</h2>
            <h3>Selecciona un Nivel de Dificultad</h3>
```

```

        </div>
        <div class="card-body text-center">
            <div class="list-group">
                <a href="{% url 'game' level='Básico' %}"
class="list-group-item list-group-item-action list-group-item-primary mb-
2">
                    Nivel Básico (6 intentos, 4x4, 60s)
                </a>
                <a href="{% url 'game' level='Medio' %}"
class="list-group-item list-group-item-action list-group-item-warning mb-
2">
                    Nivel Medio (4 intentos, 4x4, 60s)
                </a>
                <a href="{% url 'game' level='Avanzado' %}"
class="list-group-item list-group-item-action list-group-item-danger mb-
2">
                    Nivel Avanzado (2 intentos, 4x4, 60s)
                </a>
            </div>
            <hr>
            <p><a href="{% url 'profile' %}" class="btn btn-
info mt-3">Ver mi Perfil</a></p>
            <p><a href="{% url 'logout' %}" class="btn btn-
secondary mt-2">Cerrar Sesión</a></p>
        </div>
    </div>
</div>
</div>
</div>
<script src="https://code.jquery.com/jquery-
3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.mi
n.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.
js"></script>
</body>
</html>

```


- **Game.js:** Implementa la mecánica visual del juego con temporizador y emparejamiento.

```
document.addEventListener('DOMContentLoaded', () => {
  const gameBoard = document.getElementById('game-board');
  const attemptsLeftSpan = document.getElementById('attempts-left');
  const timeLeftSpan = document.getElementById('time-left');
  const victorySound = document.getElementById('victory-sound');
  const defeatSound = document.getElementById('defeat-sound');
  const backgroundMusic = document.getElementById('background-music');

  // Mapeo de niveles a nombres de archivo de música de fondo (sin la
  ruta estática aún)
  // gameLevel se pasa desde el contexto de Django en game.html
  const backgroundMusicFileNames = {
    'Básico': 'background_music_basic.mp3',
    'Medio': 'background_music_medium.mp3',
    'Avanzado': 'background_music_advanced.mp3'
  };

  //profile.total_wins

  let attempts = parseInt(attemptsLeftSpan.textContent);
  let timeLeft = parseInt(timeLeftSpan.textContent);
  let timer;
  let matchedPairs = 0;
  // Asegúrate de que cardsData tenga datos; si no, totalPairs será 0 y
  el juego terminará inmediatamente.
  const totalPairs = cardsData.length / 2;

  let flippedCards = [];
  let lockBoard = false; // Para evitar clics durante la espera de
  comparación
  backgroundMusic.volume = 0.4; // Adjust volume as needed
  backgroundMusic.play().catch(e => {
    console.log("Música de fondo no se pudo reproducir
    automáticamente. Permiso del usuario requerido.", e);
  });

  // Función para crear el tablero
  function createBoard() {
    // Asegúrate de que cardsData no esté vacía
    if (cardsData && cardsData.length > 0) {
      cardsData.forEach((cardValue, index) => {
        const cardElement = document.createElement('div');
        cardElement.classList.add('card'); // Añade la clase
        'card' de CSS
      });
    }
  }
});
```

```

        cardElement.dataset.cardValue = cardValue; // Valor para
comparación
        cardElement.dataset.index = index; // Índice único para
la carta

        cardElement.innerHTML = `
            <div class="card-inner">
                <div class="card-front"></div>
                <div class="card-back">${cardValue}</div>
            </div>
        `;
        cardElement.addEventListener('click', flipCard);
        gameBoard.appendChild(cardElement);
    });
} else {
    console.error("No hay datos de cartas para crear el tablero.
Asegúrate de que 'cards' se esté pasando correctamente desde Django.");
    gameBoard.innerHTML = "<p>No se pudieron cargar las cartas.
Por favor, intente de nuevo.</p>";
}
}

function flipCard() {
    if (lockBoard) return; // Si el tablero está bloqueado, no hacer
nada
    if (this === flippedCards[0]) return; // Evita voltear la misma
carta dos veces

    this.classList.add('flipped'); // Añade la clase para voltear
visualmente
    flippedCards.push(this);

    if (flippedCards.length === 2) {
        lockBoard = true; // Bloquea el tablero mientras se comparan
las cartas
        checkForMatch();
    }
}

function checkForMatch() {
    const [firstCard, secondCard] = flippedCards;
    const isMatch = firstCard.dataset.cardValue ===
secondCard.dataset.cardValue;

    if (isMatch) {
        firstCard.classList.add('matched'); // Marca como emparejada
        secondCard.classList.add('matched');
        disableCards(); // Deshabilita las cartas emparejadas
        matchedPairs++; // Incrementa el contador de pares
    }
}

```

```

        if (matchedPairs === totalPairs) {
            endGame(true); // Todas las cartas emparejadas, el
jugador gana
        }
    } else {
        attempts--; // No es un match, decrementa intentos
        attemptsLeftSpan.textContent = attempts;
        if (attempts <= 0) {
            endGame(false); // No quedan intentos, el jugador pierde
        } else {
            unflipCards(); // Voltea las cartas de nuevo
        }
    }
}

function disableCards() {
    // Elimina los event listeners para que las cartas emparejadas no
se puedan voltear de nuevo
    flippedCards.forEach(card => card.removeEventListener('click',
flipCard));
    resetBoard();
}

function unflipCards() {
    setTimeout(() => {
        // Voltea las cartas de nuevo si no son un match
        flippedCards.forEach(card =>
card.classList.remove('flipped'));
        resetBoard();
    }, 1000); // Espera 1 segundo antes de voltear de nuevo
}

function resetBoard() {
    // Reinicia las cartas volteadas y desbloquea el tablero
    flippedCards = [];
    lockBoard = false;
}

function startTimer() {
    timer = setInterval(() => {
        timeLeft--;
        timeLeftSpan.textContent = timeLeft;
        if (timeLeft <= 0) {
            clearInterval(timer);
            endGame(false); // Tiempo agotado, el jugador pierde
        }
    }, 1000);
}

```

```

function endGame(isWon) {
  clearInterval(timer); // Detiene el temporizador
  lockBoard = true; // Bloquea más interacciones con el tablero
  backgroundMusic.pause(); // Detiene la música de fondo
  backgroundMusic.currentTime = 0; // Reinicia la música para la
próxima vez

  if (isWon) {
    victorySound.play();
    alert('¡Felicidades, ganaste!');
  } else {
    defeatSound.play();
    alert('¡Lo siento, perdiste!');
  }

  // Calcula la duración del juego
  // Accede al valor de initial_time_limit desde el HTML
(game.html)
  const initialTimeLimit =
parseInt(timeLeftSpan.dataset.initialTimeLimit);
  //const initialTimeLimit =
parseInt(document.getElementById('time-left').dataset.initialTimeLimit ||
timeLeft);
  const finalTimeLeft = parseInt(timeLeftSpan.textContent);
  let duration;
  if (isWon) {
    duration = initialTimeLimit - finalTimeLeft;
  } else {
    duration = initialTimeLimit - Math.max(0, finalTimeLeft);
  }
  if (duration < 0) duration = initialTimeLimit; // Ensure duration
is not negative

  // Envía los resultados al backend
fetch(`/game/end/${gameSessionId}/`, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-CSRFToken': csrfToken // Token CSRF para seguridad en
Django
  },
  body: JSON.stringify({
    is_won: isWon, // This should be a boolean (true/false)
    duration: duration // This should be a number
  })
})
.then(response => {
  if (!response.ok) {

```

```

        return response.json().then(err => { throw new
Error(`Error HTTP ${response.status}: ${JSON.stringify(err)}`); });
    }
    return response.json();
  })
  .then(data => {
    console.log('Resultados guardados:', data);
    // Redirige al perfil o a la selección de nivel después de un
breve retraso
    setTimeout(() => window.location.href = profileUrl, 3000); //
¡Aquí se usa la nueva variable!
  })
  .catch(error => console.error('Error al guardar resultados:',
error));
}

// Inicializar el juego al cargar la página
createBoard();
startTimer();
});

```

-Style.css: es un archivo de hojas de estilo en cascada (CSS) que se utiliza para definir la presentación y el diseño visual de las páginas web.

Define cómo se ve el juego. Controla los colores, las fuentes, el tamaño de las cartas, las animaciones de volteo y el diseño general. Sin style.css, el juego se vería como texto y enlaces sin formato, haciendo la experiencia de usuario mucho menos atractiva.

```

body {
  background-color: #f8f9fa; /* Color de fondo claro */
  font-family: 'Arial', sans-serif; /* Fuente estándar */
  margin: 0;
  padding: 20px;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh; /* Altura mínima de la ventana */
}

/* Estilo para el contenedor principal de la página */
.container {
  background-color: #ffffff; /* Fondo blanco */
  padding: 30px;
}

```

```

border-radius: 10px; /* Bordes redondeados */
box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1); /* Sombra sutil */
max-width: 900px; /* Ancho máximo para el contenido */
width: 100%;
}

h1, h2, h3 {
color: #343a40; /* Color oscuro para los títulos */
text-align: center;
margin-bottom: 20px;
}

p {
text-align: center;
}

/* Estilos de botones Bootstrap personalizados */
.btn-primary, .btn-success, .btn-info, .btn-secondary {
margin-top: 10px;
}

/* Estilos específicos para las tarjetas del juego */
#game-board {
display: grid;
/* Grid de 4x4, ajusta el tamaño de las columnas automáticamente */
grid-template-columns: repeat(4, 1fr);
gap: 15px; /* Espacio entre las tarjetas */
max-width: 550px; /* Ancho máximo del tablero */
margin: 20px auto; /* Centrar el tablero */
padding: 10px;
border: 1px solid #e0e0e0;
border-radius: 8px;
background-color: #f0f0f0;
}

.card {
background-color: #007bff; /* Color del reverso de la carta */
border: 3px solid #0056b3; /* Borde más pronunciado */
height: 120px; /* Altura fija para las cartas */
border-radius: 8px;
cursor: pointer;
perspective: 1000px; /* Para la animación 3D de volteo */
position: relative;
display: flex;
justify-content: center;
align-items: center;
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
transition: transform 0.3s ease; /* Suave transición al voltear */
}

```

```
.card:hover {
    transform: scale(1.03); /* Ligeramente más grande al pasar el ratón */
}

.card-inner {
    position: absolute;
    width: 100%;
    height: 100%;
    text-align: center;
    transition: transform 0.6s; /* Velocidad de la animación de volteo */
    transform-style: preserve-3d; /* Permite la rotación 3D */
}

.card.flipped .card-inner {
    transform: rotateY(180deg); /* Gira la carta para mostrar el anverso */
}

.card-front, .card-back {
    position: absolute;
    width: 100%;
    height: 100%;
    -webkit-backface-visibility: hidden; /* Oculta la parte trasera de la carta */
    backface-visibility: hidden;
    border-radius: 8px;
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 2.5em; /* Tamaño grande para el contenido de la carta */
    font-weight: bold;
    color: #fff;
    user-select: none; /* Evita que el texto de la carta se seleccione */
}

.card-front {
    background-color: #007bff; /* Color del reverso (azul) */
    border: 2px solid #0056b3;
}

.card-back {
    background-color: #28a745; /* Color del anverso (verde) */
    border: 2px solid #1e7e34;
    transform: rotateY(180deg); /* Posiciona el anverso detrás del reverso */
    color: #fff;
}
```

```

/* Estilo para las tarjetas que ya han sido emparejadas y deshabilitadas
*/
.card.matched {
  opacity: 0.5; /* Se desvanece ligeramente */
  cursor: default;
  pointer-events: none; /* No se puede hacer clic */
}

/* Mensajes de estado del juego */
.badge {
  font-size: 1.1em;
  padding: 0.6em 1em;
  margin: 0 5px;
}

/* Estilos para el formulario (login/registro) */
form p {
  margin-bottom: 1rem;
  text-align: left; /* Alinea los campos a la izquierda */
}

form label {
  display: block;
  margin-bottom: 0.5rem;
  font-weight: bold;
}

form input[type="text"],
form input[type="password"],
form input[type="email"] {
  width: 100%;
  padding: 0.75rem;
  border: 1px solid #ced4da;
  border-radius: 0.25rem;
  box-sizing: border-box; /* Incluye padding y borde en el ancho total
*/
}

/* Ocultar el control de los reproductores de audio */
audio {
  display: none;
}

```

- **Templates:** Plantillas HTML para cada pantalla del usuario.

Capturas del Funcionamiento del Juego

Ventana 1: se inicia sección con el usuario y contraseña para acceder al juego.

Iniciar Sesión

Username:

Password:

[Iniciar Sesión](#)

[¿No tienes cuenta? Regístrate aquí](#)

Ventana 2: en esta parte elegimos el nivel en cual queremos jugar.

Hola, Isabel!

Selecciona un Nivel de Dificultad

Nivel Básico (10 intentos, 8 cartas, 60s)

Nivel Medio (8 intentos, 12 cartas, 90s)

Nivel Avanzado (6 intentos, 16 cartas, 120s)

[Ver mi Perfil](#)[Cerrar Sesión](#)

Ventana 3: se visualizan las estadísticas cuando se ha ganado y se ha perdido.

Perfil de lasbel

Estadísticas Generales:

Victorias Totales: 1

Derrotas Totales: 0

Partidas Jugadas: 1

Nivel más jugado: N/A

Historial de Partidas:

Fecha/Hora	Nivel	Duración (s)	Resultado
04 Aug 2025 02:26	Básico	13.00	Victoria

Volver a Selección de Nivel

Cerrar Sesión

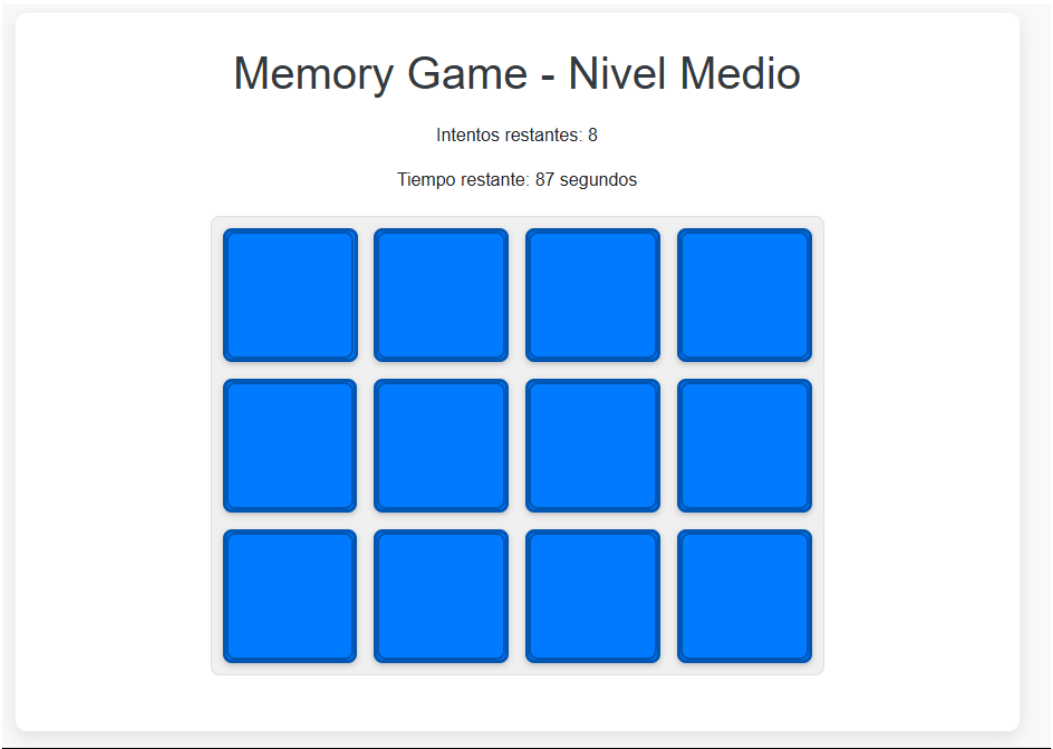
Ventana 4: en esta ventana se pude visualizar al momento de empezar el juego en el nivel básico, muestra la cantidad de intentos disponibles y el tiempo restante para que termine la partida.

Memory Game - Nivel Básico

Intentos restantes: 10

Tiempo restante: 58 segundos

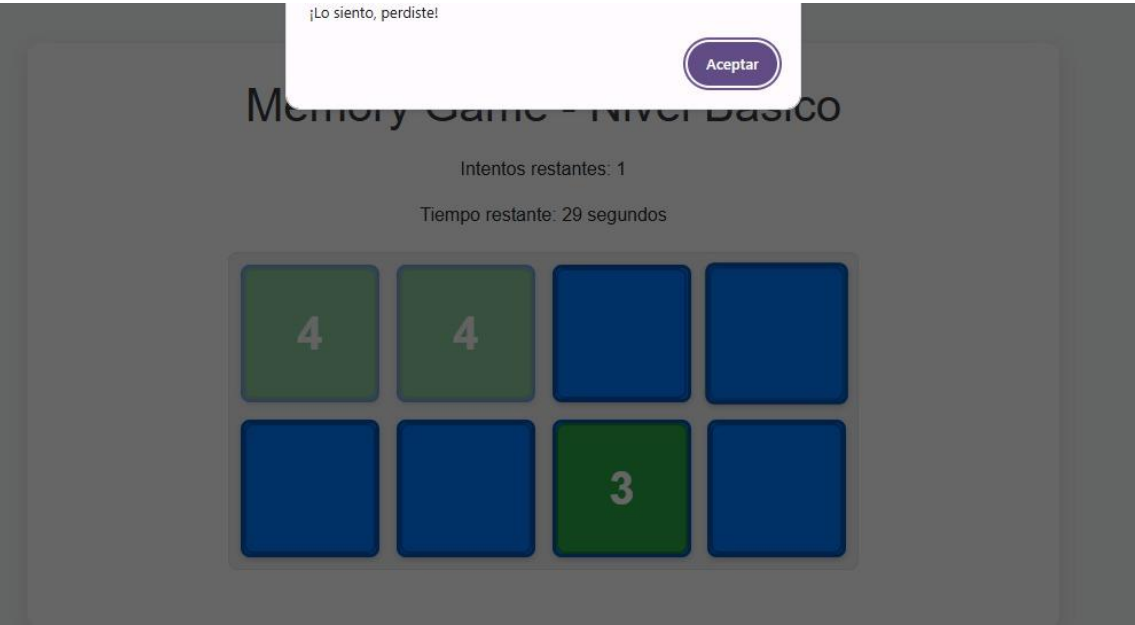
Ventana 5: en esta ventana se pude visualizar al momento de empezar el juego en el nivel Medio, muestra una cantidad de 12 cartas porque sube su nivel de dificultad.



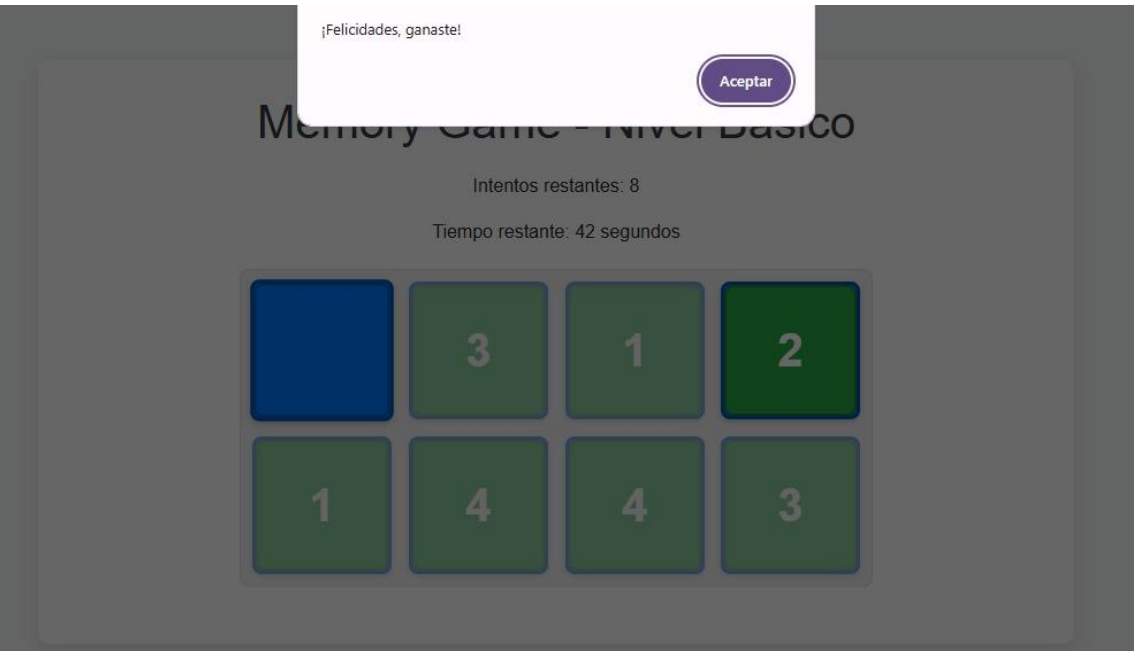
Ventana 6: en esta ventana es notoria la diferencia a la anterior ya fue este nivel es avanzado por lo tanto tiene mayor número de cartas.



Ventana 7: se observa una partida que se perdió.



Venta 8: se observa una partida que se ganó.



Ventaja 9: se observa el historial de partidas tanto de las ganadas como perdidas y también el nivel en el que se jugaron.

Historial de Partidas:			
Fecha/Hora	Nivel	Duración (s)	Resultado
04 Aug 2025 13:25	Básico	20.00	Victoria
04 Aug 2025 13:11	Básico	31.00	Derrota

Conclusiones Técnicas

- La decisión de usar Django + Docker fue basada en portabilidad, eficiencia y aislamiento.
- Se corrigió una falla inicial en la actualización de estadísticas al validar correctamente sesiones completas.
- Se evaluaron alternativas como Flask, pero se eligió Django por su robustez y sistema de autenticación incorporado.
- El uso de Docker demostró en la práctica cómo se gestiona la virtualización de recursos a nivel de contenedor.
- Este proyecto integró conceptos claves de arquitectura de computadoras con desarrollo de software moderno, proporcionando una experiencia de aprendizaje concreta y aplicable en la industria.