Manual de Módulos de Reconocimiento y Testeo para Automatización de eFootball

# 1. Introducción

Este sistema está diseñado para permitir la automatización de acciones dentro del juego eFootball mediante el reconocimiento de la pantalla actual. Se compone de tres módulos principales de Python y varios archivos de configuración JSON:

* **template\_manager\_gui.py**: Herramienta gráfica para crear y gestionar las plantillas visuales (imágenes de referencia) y definir las regiones específicas donde se aplicará el Reconocimiento Óptico de Caracteres (OCR).
* **screen\_recognizer.py**: El núcleo del sistema. Contiene la lógica para capturar la pantalla, comparar la captura con las plantillas visuales, y usar OCR como método secundario (fallback) para identificar el estado actual del juego. Utiliza optimizaciones como contexto de estado y Regiones de Interés (ROI).
* **screen\_tester\_gui.py**: Herramienta gráfica interactiva para probar el ScreenRecognizer en tiempo real mientras juegas. Permite validar las detecciones, registrar correcciones, refinar las regiones OCR (añadiendo texto esperado) y definir ROIs.

El objetivo es crear un sistema robusto que pueda identificar de forma fiable en qué pantalla del juego se encuentra el usuario para que el script de automatización principal pueda tomar las decisiones correctas.

# 2. Archivos de Configuración JSON (en la carpeta config/)

Estos archivos son cruciales para el funcionamiento del sistema. Deben estar correctamente estructurados.

**2.1. templates\_mapping.json**

* **Propósito:** Asocia un nombre descriptivo (el "estado") a una o más imágenes de plantilla (archivos .png) que representan visualmente ese estado.
* **Estructura:** Un diccionario JSON.
  + **Claves (Keys):** string. Nombres únicos y descriptivos para cada estado/pantalla del juego (ej., "menu\_principal\_home\_seleccionado", "partido\_jugando\_final").
  + **Valores (Values):** list de string. Una lista que contiene los nombres de archivo (con extensión .png) de las imágenes guardadas en la carpeta images/ que corresponden a esa clave (estado). Se pueden tener múltiples imágenes para un mismo estado si este tiene variaciones visuales significativas que quieres capturar.
* **Ejemplo:**

{

"pantalla\_bienvenida": [

"pantalla\_bienvenida\_20250403\_172226.png"

],

"menu\_principal\_home\_seleccionado": [

"menu\_principal\_home\_seleccionado\_20250403\_172940.png",

"menu\_principal\_home\_seleccionado\_20250411\_192123.png"

],

"menu\_home\_contrato": [

"menu\_home\_contrato\_20250403\_181042.png"

]

}

* **Eficiencia/Buenas Prácticas:**
  + Usa nombres de estado claros y consistentes.
  + Añade múltiples imágenes a un estado *solo si son visualmente distintas* y ayudan a reconocer variaciones (ej., un banner diferente en el menú principal). Evita redundancias.
  + La GUI template\_manager añade automáticamente un timestamp al guardar nuevas imágenes, ayudando a diferenciar capturas para el mismo nombre base.

**2.2. ocr\_regions.json**

* **Propósito:** Define áreas específicas (rectángulos) dentro de ciertas pantallas donde el sistema debe intentar leer texto usando OCR, *especialmente* cuando el reconocimiento visual por plantilla falla o necesita confirmación. También almacena el texto que *esperamos* encontrar en esas regiones para validar la detección OCR.
* **Estructura:** Un diccionario JSON.
  + **Claves (Keys):** string. Nombres de estado que coinciden exactamente con las claves de templates\_mapping.json. Solo se incluyen los estados para los que quieres definir zonas OCR.
  + **Valores (Values):** list de dict. Una lista donde cada elemento es un diccionario que representa una zona OCR para ese estado. Cada diccionario de zona tiene:
    - "region": dict. Contiene las coordenadas absolutas del rectángulo en la pantalla completa: "left", "top", "width", "height" (todos int).
    - "expected\_text": list de string. Una lista con uno o más textos que se consideran válidos si son leídos por el OCR en esa región específica. La comparación suele ser insensible a mayúsculas/minúsculas. Si la lista está vacía [], significa que se ha definido la región pero aún no se ha validado ningún texto esperado para ella.
* **Ejemplo:**

{

"menu\_principal\_home\_seleccionado": [

{

"region": {"left": 951, "top": 107, "width": 247, "height": 113},

"expected\_text": ["Home"]

}

],

"menu\_raquel\_fichado2": [

{

"region": {"left": 1100, "top": 800, "width": 1600, "height": 100},

"expected\_text": ["Has recibido Raquel Lombardi"]

},

{

"region": {"left": 1700, "top": 1150, "width": 400, "height": 150},

"expected\_text": ["OK"]

}

]

}

* **Eficiencia/Buenas Prácticas:**
  + Define regiones OCR solo donde sea necesario (pop-ups, texto variable que confirma un estado, etc.).
  + Haz las regiones lo más pequeñas posible pero que contengan todo el texto relevante para acelerar el OCR.
  + Usa el screen\_tester\_gui.py para añadir y validar los "expected\_text". Esto es crucial para la precisión del fallback OCR.

**2.3. state\_transitions.json**

* **Propósito:** Optimizar la velocidad de reconocimiento indicando qué estados son más probables que aparezcan después de un estado conocido. Limita la búsqueda inicial de plantillas.
* **Estructura:** Un diccionario JSON.
  + **Claves (Keys):** string. Nombre de un estado (el estado *anterior*).
  + **Valores (Values):** list de string. Lista de nombres de estado que son destinos *directos y probables* desde el estado clave (mediante una acción simple como mover cursor, pulsar A o B).
* **Ejemplo:** (Simplificado)

{

"menu\_principal\_home\_seleccionado": [

"menu\_principal\_misiones\_sel",

"menu\_principal\_tienda\_sel",

"menu\_home\_partido",

"menu\_home\_contrato"

],

"menu\_contrato\_jugadores\_normales": [

"menu\_contrato\_jugadores\_lista",

"menu\_principal\_contrato\_sel"

]

}

* **Eficiencia/Buenas Prácticas:**
  + Define transiciones para los flujos más comunes y desde los menús principales.
  + Incluye la posibilidad de volver atrás (ej., desde un submenú al menú principal).
  + No es necesario que sea exhaustivo para todos los estados, el sistema probará todos si no hay contexto.
  + Mantén la lista actualizada si añades nuevos estados relevantes.

**2.4. state\_rois.json**

* **Propósito:** Optimizar la velocidad del *template matching* para ciertos estados, indicando que sus plantillas visuales deben buscarse solo dentro de un rectángulo específico (Región de Interés - ROI) en lugar de en toda la pantalla. Ideal para elementos fijos.
* **Estructura:** Un diccionario JSON.
  + **Claves (Keys):** string. Nombre del estado al que se aplica el ROI.
  + **Valores (Values):** dict. Un diccionario con las coordenadas absolutas del ROI: "left", "top", "width", "height" (todos int).
* **Ejemplo:**

{

"menu\_principal\_home\_seleccionado": {"left": 50, "top": 45, "width": 1000, "height": 160},

"menu\_principal\_misiones\_sel": {"left": 50, "top": 45, "width": 1000, "height": 160},

"pantalla\_bienvenida": {"left": 1400, "top": 1900, "width": 1050, "height": 200}

}

* **Eficiencia/Buenas Prácticas:**
  + Define ROIs solo para estados donde la plantilla visual clave esté en una posición relativamente fija.
  + Usa el screen\_tester\_gui.py para definir visualmente estos ROIs sobre una captura de pantalla.
  + Asegúrate de que el ROI sea lo suficientemente grande para contener la plantilla con un pequeño margen, pero no innecesariamente extenso.

# 3. Módulo: template\_manager\_gui.py

* **Propósito Principal:** Crear y gestionar las plantillas visuales (templates\_mapping.json y archivos PNG en images/) y definir las regiones OCR iniciales (ocr\_regions.json, sin texto esperado inicialmente, o añadiéndolo si se desea).
* **Características Clave:**
  + **Captura:** Permite capturar pantalla completa o una región de un monitor específico.
  + **Guardar Nueva Plantilla:** Guarda la imagen capturada en images/ con un nombre base y timestamp. Actualiza (o crea la entrada) en templates\_mapping.json añadiendo el nuevo nombre de archivo a la lista del nombre base.
  + **Seleccionar Existente:** Muestra una lista desplegable (Combobox) con los nombres de estado de templates\_mapping.json (ordenados).
  + **Cargar/Previsualizar:** Carga y muestra la primera imagen asociada a la plantilla seleccionada (o una cargada manualmente). Permite redimensionar la ventana.
  + **Marcar Región OCR:** Abre una ventana para dibujar un rectángulo sobre la previsualización.
  + **Texto Esperado OCR (Opcional):** Permite introducir texto(s) esperado(s) (separados por '|') *antes* de marcar una región OCR.
  + **Guardar Zonas OCR:** Guarda la lista de regiones marcadas (cada una como {"region": {...}, "expected\_text": [...]}) en ocr\_regions.json bajo el nombre de la plantilla seleccionada en el Combobox. Sobrescribe las regiones anteriores para esa plantilla.
  + **Limpiar Zonas:** Borra las zonas OCR marcadas *actualmente* en la GUI (no en el archivo).
  + **Refrescar Lista:** Vuelve a cargar los nombres de templates\_mapping.json en el Combobox.
* **Uso Típico:**
  1. **Para Nueva Plantilla:** Captura -> Escribe Nombre Base -> Guarda Nueva Plantilla.
  2. **Para Definir Zonas OCR:** Selecciona Plantilla Existente -> (Opcional: Escribe Texto Esperado) -> Marca Región OCR (repite si es necesario) -> Guarda Zonas OCR.
* **Archivos Modificados:** images/\*.png, config/templates\_mapping.json, config/ocr\_regions.json.

# 4. Módulo: screen\_recognizer.py

* **Propósito Principal:** Identificar el estado actual de la pantalla del juego usando la información de los archivos JSON y las imágenes.
* **Lógica Clave:**
  1. **Carga Inicial:** Carga todos los mappings JSON y las imágenes de plantilla en memoria (\_load\_all\_data, \_load\_templates).
  2. **Captura:** Captura la pantalla completa del monitor especificado (capture\_screen).
  3. **Contexto:** Determina qué estados probar primero basándose en last\_recognized\_state y state\_transitions.json.
  4. **ROI:** Para cada estado a probar, verifica si hay un ROI definido en state\_rois.json.
  5. **Template Matching:** Compara las plantillas del estado actual contra la pantalla completa o el ROI (find\_template\_on\_screen). Prioriza los estados definidos por el contexto.
  6. **Decisión Template:** Si encuentra una coincidencia por encima del threshold, devuelve ese estado. Guarda los estados con coincidencias parciales (entre ocr\_fallback\_threshold y threshold).
  7. **OCR Fallback (si no hubo match claro):** Itera sobre los estados con coincidencias parciales (ordenados por confianza). Para cada uno:
     + Busca sus regiones definidas en ocr\_regions\_mapping.
     + Para cada región, captura esa pequeña área, extrae texto (\_extract\_and\_clean\_text).
     + **Verifica el Texto:** Compara el texto extraído con la lista expected\_text asociada a esa región en el JSON.
     + **Decisión OCR:** Si *alguna* región de un estado candidato coincide con su texto esperado, devuelve ese estado.
  8. **Resultado Final:** Devuelve el estado detectado ('template' o 'ocr') o 'unknown'. Actualiza last\_recognized\_state.
* **Métodos Importantes:**
  1. \_\_init\_\_(...): Inicializa y carga datos.
  2. reload\_data(): Vuelve a cargar todos los archivos JSON y plantillas.
  3. recognize\_screen\_for\_test(): Ejecuta el proceso completo de reconocimiento y devuelve resultados detallados para el tester.
* **Archivos Leídos:** config/templates\_mapping.json, config/ocr\_regions.json, config/state\_transitions.json, config/state\_rois.json, images/\*.png.

# 5. Módulo: screen\_tester\_gui.py

* **Propósito Principal:** Probar interactivamente el ScreenRecognizer, validar sus resultados, refinar los datos OCR (añadiendo texto esperado) y definir ROIs.
* **Características Clave:**
  + **Reconocer:** Botón para ejecutar recognizer.recognize\_screen\_for\_test() y mostrar los resultados (método, estado, confianza, tiempo).
  + **Validar:** Botones "Confirmar" y "Negar" para indicar si la detección fue correcta. Loggea esta acción.
  + **Corrección:** Si se niega o falla la detección, muestra un Combobox (ordenado) con todos los estados conocidos para seleccionar el correcto y un botón para loggear esta corrección.
  + **Detalles OCR:** Si la detección fue por OCR, muestra una tabla (Treeview) con cada región probada, el texto extraído, el texto esperado (si existe) y si coincidieron.
  + **Edición/Confirmación OCR:** Permite:
    - Seleccionar una o más filas en la tabla OCR.
    - Pulsar "Confirmar Texto Extraído": Añade el texto *realmente extraído* por el OCR a la lista expected\_text de las regiones seleccionadas en ocr\_regions.json.
    - Escribir texto(s) esperado(s) (separados por '|') y pulsar "Guardar Texto Editado": *Reemplaza* la lista expected\_text de las regiones seleccionadas con el texto introducido.
  + **Definir ROI:** Botón (activo si se detectó algo o se negó) que captura la pantalla actual y abre una ventana para dibujar el ROI para el estado detectado/corregido, guardándolo en state\_rois.json.
  + **Abrir Gestor:** Botón (activo si se negó o falló la detección) para lanzar template\_manager\_gui.py en otro proceso.
  + **Recargar Datos:** Botón para llamar a recognizer.reload\_data(), útil después de hacer cambios en el Gestor.
  + **Logging:** Registra todas las acciones y resultados en tester\_log.log.
* **Uso Típico (Ciclo de Refinamiento):**
  1. Ejecuta el juego y el tester.
  2. Navega a una pantalla en el juego.
  3. Pulsa "Reconocer" en el tester.
  4. **Si es Correcto:** Pulsa "Confirmar". Si fue OCR y quieres guardar el texto extraído como válido, selecciónalo en la tabla y pulsa "Confirmar Texto Extraído". Si quieres definir un ROI, pulsa "Definir/Editar ROI".
  5. **Si es Incorrecto:** Pulsa "Negar".
     + Selecciona el estado correcto en el Combobox de corrección y pulsa "Registrar Corrección".
     + Opcional: Pulsa "Definir/Editar ROI" para ajustar el ROI del estado que *debería* haberse detectado (el corregido).
     + Opcional: Pulsa "Abrir Gestor" para ir a la otra GUI, capturar una nueva imagen para el estado correcto o mejorar las existentes. Después, vuelve al tester y pulsa "Recargar Datos Reconocedor".
  6. **Si fue OCR Incorrecto:** Revisa la tabla OCR. Escribe el texto correcto en el campo de edición, selecciona la(s) fila(s) correspondiente(s) y pulsa "Guardar Texto Editado".
  7. Repite el proceso para diferentes pantallas y flujos.
* **Archivos Modificados:** config/ocr\_regions.json (al confirmar/editar texto esperado), config/state\_rois.json (al definir/editar ROI).
* **Archivos Leídos:** Usa ScreenRecognizer que lee todos los JSON e imágenes. Escribe en tester\_log.log.

# 6. Flujo de Trabajo General y Buenas Prácticas

1. **Captura Inicial (Template Manager):** Usa template\_manager\_gui.py para capturar las imágenes iniciales de las pantallas clave y darles nombres descriptivos. Guarda las plantillas.
2. **Definición Inicial OCR/ROI (Opcional - Template Manager):** Puedes usar el template\_manager para definir algunas zonas OCR básicas o ROIs si ya conoces las coordenadas.
3. **Testeo y Refinamiento (Tester):** Usa screen\_tester\_gui.py como herramienta principal.
   * Navega por el juego y usa "Reconocer".
   * Valida las detecciones correctas ("Confirmar").
   * Cuando falle ("Negar" o "unknown"):
     + Registra el estado correcto ("Registrar Corrección").
     + Usa "Definir/Editar ROI" para ajustar el área de búsqueda si aplica.
     + Si el problema es la plantilla visual, usa "Abrir Gestor", captura una mejor imagen para ese estado (se añadirá al mapping), vuelve al tester y "Recargar Datos".
     + Si el problema fue OCR (detectó el estado correcto pero por OCR y quieres validarlo, o detectó un estado incorrecto por OCR): Usa la tabla y los botones OCR en el tester para "Confirmar Texto Extraído" o "Guardar Texto Editado".
4. **Iteración:** Repite el testeo en diferentes condiciones y flujos del juego. Revisa el tester\_log.log para identificar patrones de error.
5. **Mantenimiento:** Mantén los archivos JSON organizados y revisa periódicamente si hay plantillas redundantes o ROIs/regiones OCR que ya no son precisas debido a actualizaciones del juego.
6. **Backups:** Haz copias de seguridad de tu carpeta config regularmente.