Universidad Autónoma de Madrid

Departamento de Informática

Proyecto de Sistemas Informáticos
Práctica - 4

Roberto Latorre

Índice

1. Objetivos

En esta práctica se implementará el juego del ratón y el gato enunciado en la práctica anterior. A continuación se enumeran los principales requisitos funcionales de la aplicación:

- Los usuarios se identifican usando un nombre de usuario y una clave.
- Se puede crear nuevos usuarios.
- Los dos jugadores de una partida pueden jugar simultáneamente conectándose desde navegadores diferentes.
- Se puede jugar más de una partida simultáneamente.
- No se permite realizar movimientos ilegales.
- Cuando uno de los jugadores ha ganado la partida, el juego se da por finalizado.
- Se guarda en base de datos registro de cada partida, incluyendo cada uno de los movimientos y quién lo realiza.
- Se puede reproducir cualquier partida almacenada en la base de datos.

2. Trabajo a realizar durante la primera semana

El trabajo a realizar en esta práctica parte del código desarrollado en la práctica anterior. Antes de nada, almacenar en un nuevo proyecto en *Github* una copia de vuestro código de la práctica 3. No reutilicéis los repositorios usados en la práctica anterior.

Esta semana nos vamos a limitar a modificar el frontal general de la aplicación para hacerlo atractivo y *responsive*. Los "templates" usados en la práctica 3 son poco atrayentes, ya que su objetivo era ayudar a definir los mecanismos de comunicación entre los distintos componentes y validar el correcto funcionamiento de la lógica de la aplicación. Uno de los principales objetivos de esta práctica es modificar estos templates o crear unos nuevos de forma que la aplicación disponga de un frontal

responsive amigable para el usuario. Al modificar los templates, gran parte los tests definidos en el fichero `tests_services.py` dejarán de ser funcionales. Esto no te debe preocupar.

Durante la primera semana te recomendamos crear las guías de estilo CSS que consideres oportunas para modificar la interfaz de usuario a tu gusto. Estas guías las puedes definir "a mano", reutilizar alguna pública o utilizar cualquier framework que conozcas. Para evitar problemas a la hora de trabajar con ficheros CSS, recuerda lo que tuviste que hacer en la práctica 2 para que éstos sean accesibles por *Django* y se pudiesen descargar correctamente al cliente tanto en el entorno de desarrollo local como en *Heroku*.

2.1. Menú de la aplicación

A diferencia del "menú" de la práctica 3, en el que aparecían todas las opciones para todo tipo de usuario, en la versión definitiva de la aplicación vamos a distinguir, al menos, dos menús. Los usuarios anónimos podrán registrarse en el sistema y autenticarse. Los usuarios autenticados podrán crear un nuevo juego, unirse como ratón a uno existente, jugar o reproducir una partida finalizada (nueva funcionalidad). Disponer de dos menús independientes no debe evitar mantener los controles de acceso especificados en la práctica anterior. De hecho, usando la clase `Counter`, se mantendrá un registro del número de veces que la aplicación devuelve un error controlado por recibir una petición no autorizada: acciones anónimas invocadas por usuarios autenticados y viceversa, usuarios que intentan jugar o reproducir una partida de la que no son jugador, etc.

2.2. Pantalla de juego

La página que más modificaciones va a sufrir respecto a la versión anterior, no solo visuales, sino también funcionales es la pantalla del juego. En esta primera semana nos vamos a limitar a cambiar el aspecto general de la pantalla "dibujando" un tablero de juego como el de la Figura 1 con celdas de color y celdas blancas por las que se moverán los gatos y los ratones.

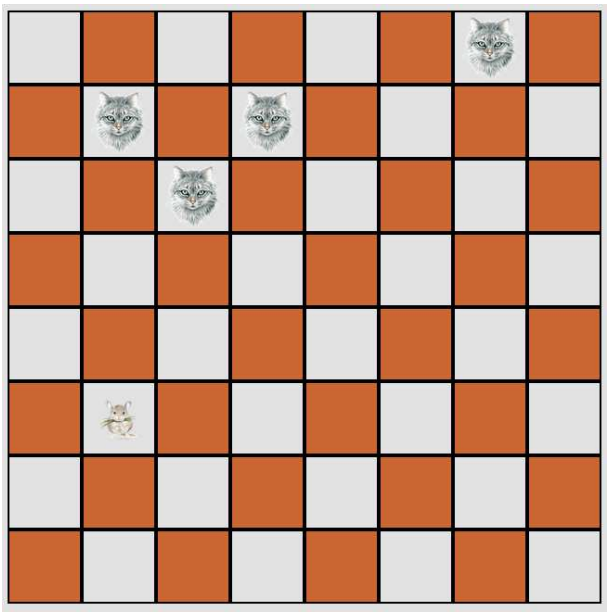

Figura 1: Ejemplo de tablero de juego

3. Trabajo a realizar hasta el final del proyecto

3.1. Requisitos pendientes

Los requisitos pendientes de desarrollar en la práctica 3 son: la finalización de partida, la selección de partidas a las que unirse/reproducir y la reproducción de partidas finalizadas.

3.1.1. Finalización de partida

Basándote en las pautas de diseño establecidas en la práctica anterior, añade la funcionalidad necesaria a las clases `Game` y/o `Move` para que, tras realizar un movimiento que haga que uno de los jugadores gane la partida, ésta se cierre automáticamente. De esta forma, el juego se dará por finalizado y no será posible realizar más movimientos por parte de ninguno de los dos jugadores.

Incluir en el fichero `tests_additional_P4.py` una clase `GameEndTests` con la batería de tests que consideres necesaria para probar completamente el correcto fun-

cionamiento de la nueva funcionalidad desarrollada. Para ello, apóyate en los tests del fichero `tests_models.py`.

3.1.2. Generalización de la selección de partidas

En la práctica 3 ya se desarrolló un servicio, `select_game_service`, que devolvía los juegos activos en los que participaba el usuario que lo invocaba, permitiendo la selección de uno de ellos para comenzar o continuar jugando la partida. En esta práctica debes generalizar este servicio para, además de la selección de la partida a jugar, permita seleccionar una partida a la que unirse como ratón o una partida que reproducir (ver más adelante).

Conceptualmente, en los tres casos el servicio hace lo mismo, la diferencia son las condiciones que deben cumplir los juegos a devolver en cada caso:

- Jugar: Juegos activos en los que el usuario es uno de sus jugadores.
- Unirse: Juegos creados por un usuario diferente que están a la espera de ratón.
- Reproducir: Juegos finalizados en los que el usuario es uno de sus jugadores.

Desde el punto de vista de su diseño, puedes utilizar cualquiera de las aproximaciones utilizadas en la práctica anterior:

- GET → muestra los juegos que cumplen los criterios de selección. POST → permite la selección de uno de ellos identificado por su id.
- Sendas peticiones GET, pero en el caso de la selección, el id del juego seleccionado se recibe como parte de una url dinámica.

En cualquier caso, sea cual sea tu elección, no olvides que las dos fases de la selección son asíncronas y en algunos casos el juego puede cambiar de estado entre ellas.

3.1.3. Reproducir partida

Para reproducir una partida, el usuario primero deberá identificar la partida a reproducir. La elección se hará sobre el conjunto de partidas ya finalizadas en las que

ha participado. Una vez seleccionada la partida, se pasará a una pantalla análoga a la del juego en la que el usuario dispondrá de un control que le permitirá:

- Realizar el siguiente movimiento.
- Rebobinar al movimiento anterior.
- Indicar que la partida se reproduzca automáticamente (play). La tasa de refresco de movimientos será de 2 segundos.
- Parar la reproducción automática.
- Una vez llegue al final de la partida, se mostrará el mensaje de finalización y el jugador ganador.

Como requisito no funcional del desarrollo de esta funcionalidad, se debe tener en cuenta que, una vez cargado el estado inicial de la partida, **todas las peticiones al servidor deben realizarse con AJAX**. Para ello:

1. Se debe tener en cuenta que en la aplicación se está utilizando la protección contra CSRF (*Cross Site Request Forgeries*) proporcionada por *Django*. Esta protección también afecta a las peticiones AJAX recibidas por el servidor, de forma que si no se especifica en la llamada un token CSRF válido, el servidor denegará el servicio, devolviendo un error 403. Para saber los pasos a seguir para obtener el token CSRF e incluirlo en las peticiones repasa la documentación de *Django* relativa a la protección CSRF (`https://docs.djangoproject.com/en/2.2/ref/csrf/`).
2. Para obtener los datos del siguiente movimiento a reproducir, desarrolla un nuevo servicio, `get_move_service` que, una vez almacenado en la sesión el identificador del juego a reproducir, reciba a través de POST si se quiere el siguiente movimiento de la secuencia de movimientos de la partida (`shift=+1`) o el anterior (`shift=-1`) y devuelva una respuesta JSON en la que al menos se develva la siguiente información:

```
{
  'origin': casilla origen del movimiento,
  'target': casilla destino del movimiento,
  'previous': indicador de si hay movimientos anteriores,
  'next': indicador de si hay movimientos posteriores
}
```

Ten en cuenta que en el caso de ir hacia atrás en la partida, el origen y el destino del movimiento se deben intercambiar. Si consideras que necesitas campos adicionales para realizar la actualización de la pantalla puedes añadir libremente los campos que consideres necesarios. Para probar el correcto funcionamiento de este servicio, se proporciona un par de tests unitarios (no definen una batería de prueba completa) en el fichero texttttests_services_P4.py.

Una vez recibida la información del servidor, la pantalla se modificará dinámicamente en el cliente moviendo el gato o el ratón según corresponda a su nueva posición.

3.2. Funcionalidad en el cliente

A continuación se enumeran una serie de funcionalidades que incorporar a la parte cliente de vuestros proyectos:

- La primera modificación a realizar para mejorar la usabilidad de la aplicación es poder manejar las piezas de juego con el ratón. El comportamiento solicitado por defecto consistirá en seleccionar la pieza que se desea mover con un click y la casilla destino con un segundo click. Como alternativa, se sugiere implementar el movimiento mediante drag & drop. En este caso, se recomienda revisar la documentación de *jQueryUI* (`https://api.jqueryui.com/`), en particular aquella relativa a los widget *draggable* y *droppable*. Una vez identificado el origen y destino del movimiento, una primera aproximación para realizar el movimiento puede consistir en rellenar y enviar el `MoveForm` correspondiente con *JavaScript*, haciendo una petición síncrona que actualice la

pantalla completa. Más adelante podrías considerar implementar el envío de datos y el procesamiento de la respuesta mediante AJAX.

- Incorporar a todos los listados de juegos un filtro por: jugador gato, jugador ratón y estado. Este filtro deberá comportarse de forma coherente con la pantalla en la que aparece. Por ejemplo, incorporar el filtro, no debería permitir a un usuario acceder a un juego del que no es jugador.

- Paginar los listados de juegos. En lugar de listar todos los juegos que cumplen un determinado criterio de búsqueda y que el usuario tenga que hacer scroll sobre ellos, dividir los listados de juegos en páginas con un número predeterminado de juegos, incluyendo controles para poder moverse por las páginas del listado. En `https://docs.djangoproject.com/en/2.2/topics/pagination/` disponéis de la documentación de las componentes que proporciona *Django* para la gestión de datos paginados.

4. Trabajo a presentar al finalizar la práctica

- Aseguraros que vuestro código satisface **todos** los tests proporcionados en `tests_models.py`, `tests_function.py` (ambos proporcionados en la práctica anterior) y `tests_services_P4.py`. No es admisible que se modifique el código de los tests.
- Implementar todos los tests que consideres necesarios para cubrir la funcionalidad desarrollada. Estos tests se deben incluir en clases dentro de un fichero llamado `tests_additional_P4.py`.
- Incluir en la raíz del proyecto un fichero llamado coverage.txt que contenga el resultado de ejecutar el comando `coverage` para cada uno de los tests.
- Desplegar y probar la aplicación en *Heroku*.
- Incluir un **breve** manual de usuario de la aplicación de no más de 5 páginas en formato pdf. El manual contendrá las instrucciones adecuadas para que los usuarios de vuestra aplicación puedan utilizarla. No se os pide un manual para el desarrollador en el cual se describa la arquitectura de la aplicación, ni una descripción sobre la arquitectura de *Django*.
- Subir a *Moodle*, en un único fichero zip, el proyecto de *Django* conteniendo la aplicación desarrollada. En concreto se debe subir el fichero obtenido ejecutando el comando `git archive --format zip --output ../assign4_final.zip master` desde el directorio del proyecto. En este punto es importante asegurarse que la variable `ALLOWED_HOSTS` del fichero `settings.py` incluido en la entrega contiene tu dirección de despliegue en *Heroku* (si no aparece no podremos corregir tu aplicación). Asimismo, comprobar que tanto el nombre de usuario como la password del usuario de administración son *alumnodb*.

5. Criterios de evaluación

Para aprobar con 5 puntos es necesario satisfacer en su totalidad los siguientes criterios:

- Todos los ficheros necesarios para ejecutar la aplicación se han entregado a tiempo.
- Se ha cambiado el frontal de la aplicación y es amigable.
- Tanto en *Heroku*, como localmente, es posible jugar una partida (esto incluye, dar de alta a los usuarios, hacer login y jugar).
- Se ha desarrollado correctamente la finalización de la partida.
- Las piezas del juego se mueven con el ratón.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 6.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- Los tests incluidos en la clase `GameEndTests` son correctos y completos.
- El sistema soporta que varias partidas se jueguen simultáneamente entre diferentes jugadores. Se asume que cada jugador se conecta usando un navegador con una sesión diferente.
- Se ha implementado correctamente con AJAX la reproducción de partidas.
- La aplicación felicita al jugador ganador.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 7.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- La aplicación es completamente *responsive*.
- La aplicación es funcionalemente correcta y en todos los casos de error se da feedback claro y explicativo al usuario (p.ej., movimientos no válidos).

- Se mantiene un registro del número de veces que la aplicación devuelve un error controlado por recibir una petición no autorizada.
- El servicio de selección de juegos se ha generalizado correctamente, existiendo un único servicio customizable y no tres servicios independientes.

Cumpliendo los siguientes criterios se puede optar a una nota máxima de 8.9:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.
- El código es legible, eficiente, está bien estructurado y comentado.
- Se utilizan las herramientas que proporciona el framework.
- Sirva como ejemplo de los puntos anteriores:
  - Las búsquedas las realiza la base de datos no se cargan todos los elementos de una tabla y se busca en las funciones definidas en `views.py`.
  - Los errores se procesan adecuadamente y se devuelven mensajes de error comprensibles.
  - El código presenta un estilo consistente y las funciones están comentadas incluyendo su autor. Nota: el autor de una función debe ser único.
  - La comprobación que el movimiento del ratón/gato es correcto es exhaustiva y no se basa el elementos "hardcodeados".
  - Se indenta consistentemente sin mezclar espacios y tabuladores.
  - Se es coherente con los criterios de estilos marcados por *Flake8*.
- Se ha incluido el manual de usuario y es correcto.
- Se han implementado los filtros de juegos y la paginación.

Para optar a la nota máxima se deben cumplir lo siguientes criterios:

- Los criterios enunciados en el párrafo anterior se satisfacen en su totalidad.

- La aplicación es robusta, respondiendo adecuadamente en todas las situaciones. Sirva de ejemplo, que no permite ejecutar funcionalidad indebidamente mediante una llamada a url, o que al solicitar una url no válida da un mensaje de error (404) integrado en la aplicación.
- Se satisfacel al menos una de las siguientes condiciones:
  - Las piezas se mueven mediante drag & drop.
  - La aplicación es SPA (*Single Page Application*).

Nota: Entrega final fuera de plazo → substraer un punto por cada día (o fracción) de retraso en la entrega.
Nota: El código usado en la corrección de la práctica será el entregado en *Moodle*. Bajo ningún concepto se usará el código existente en *Heroku*, *Github* o cualquier otro repositorio.