

# *Megaman Inicia*

## *Ejercicio Final*

<b>Objetivos</b>	<ul style="list-style-type: none"><li>• Afianzar los conocimientos adquiridos durante la cursada.</li><li>• Poner en práctica la coordinación de tareas dentro de un grupo de trabajo.</li><li>• Realizar un aplicativo de complejidad media con niveles aceptables de calidad y usabilidad.</li></ul>
<b>Instancias de Entrega</b>	<b>Pre-Entrega:</b> clase 14 (07/06/2016). <b>Entrega:</b> clase 16 (21/06/2016).
<b>Temas de Repaso</b>	<ul style="list-style-type: none"><li>• Aplicaciones Cliente-Servidor multi-threading.</li><li>• Interfaces gráficas con <i>gtkmm</i></li><li>• Manejo de errores en C++</li></ul>
<b>Criterios de Evaluación</b>	<ul style="list-style-type: none"><li>• Criterios de ejercicios anteriores.</li><li>• Construcción de un sistema Cliente-Servidor de complejidad media.</li><li>• Empleo de buenas prácticas de programación en C++.</li><li>• Coordinación de trabajo grupal.</li><li>• Planificación y distribución de tareas para cumplir con los plazos de entrega pautados.</li><li>• Cumplimiento de todos los requerimientos técnicos y funcionales.</li><li>• Facilidad de instalación y ejecución del sistema final.</li><li>• Calidad de la documentación técnica y manuales entregados.</li><li>• Buena presentación del trabajo práctico y cumplimiento de las normas de entrega establecidas por la cátedra (revisar criterios en sitio de la materia).</li></ul>

## Índice

[Introducción](#)

[Personajes](#)

[Jefes](#)

[Secuaces](#)

[Obstáculos y Escenario](#)

[Dinámica de Juego](#)

[Partidas y Jugadores](#)

[Poderes Especiales](#)

[Premios o power-up](#)

[Aplicaciones Requeridas](#)

[Servidor](#)

[Física y detección de colisiones:](#)

[Logging](#)

[Configuración](#)

[Cliente](#)

[Imágenes](#)

[Editor de Niveles](#)

[Distribución de Tareas Propuesta](#)

[Restricciones](#)

[Referencias](#)

## Introducción

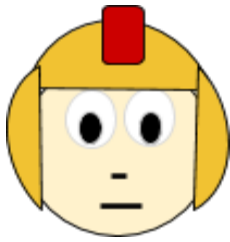
Año 20XX. El uso de robots en tareas domésticas tuvo tal incremento que es más frecuente encontrar robots en las calles que seres humanos. Las ciudades son más grandes que nunca y su contaminación afecta física y emocionalmente a los humanos. Uno de los más brillantes científicos y diseñadores de robots, el Dr. Wily, enloquece. Sus delirios de grandeza crecen de la mano de sus experimentos y del gran poder que poseen los robots que construye. Planea entonces una rebelión de robots comandados por él mismo para dominar la Tierra. Con modificaciones en 5 poderosos robots, consigue dominar los vecindarios más importantes de la Ciudad Capital y ahora es solo cuestión de tiempo para que pueda extender sus dominios.

Uno de sus colegas, el Dr. Light, viendo lo inminente que se encuentra el planeta de una guerra a gran escala decide evitarlo. Modifica a sus robots hogareños para que puedan luchar contra Willy y sus creaciones... se trata de los inicios de Megaman.

## Personajes

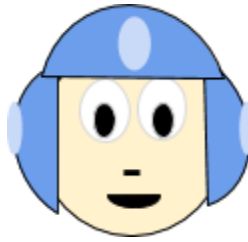
### Jefes

Tanto Megaman como los 5 robots modificados por Wily poseen formas humanas y características especiales. A continuación se detallan sus principales cualidades.



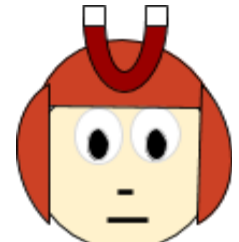
### Bombman

Modificado por el Dr. Wily. Tiene la capacidad de arrojar bombas que causan un gran daño al explotar. Sus saltos son enormes pero no es muy rápido al correr.



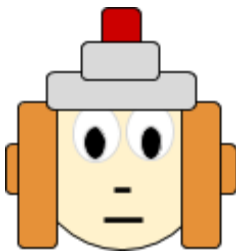
### Megaman

Robot modificado por el Dr. Light. Posee un cañón en el brazo capaz de disparar plasma. Debe destruir tantos enemigos como sea posible para detener la rebelión de robots y salvar a la Tierra. Al iniciar la partida, cada jugador posee un Megaman con 3 vidas y un tanque de energía que puede ser recargado.



### Magnetman

Modificado por el Dr. Wily. Dispara magnetos en línea recta aunque puede modificar su rumbo para dirigirlos al enemigo. No suele saltar mucho pero al hacerlo nada le impide disparar.



### Sparkman

Modificado por el Dr. Wily. Genera grandes chispas que arroja en línea recta de forma horizontal y oblicua. No es muy veloz pero salta constantemente.



### Ringman

Modificado por el Dr. Wily. Es capaz de arrojar anillos de gran tamaño que rebotan contra las paredes. Puede saltar muy alto, aunque no siempre lo hace.



### Fireman

Modificado por el Dr. Wily. Tiene la habilidad de arrojar largas llamaradas de fuego en línea recta. Es muy veloz tanto para correr como en sus saltos.

## Secuaces

En los distintos niveles, los 5 robots dispusieron a sus robots colaboradores para evitar que alguien llegue a su cámara. Los secuaces no son tan fuertes como los jefes y tienen movimientos más acotados. A continuación se detallan los tipos de enemigos que colaboran con los jefes:






### Met

**Descripción:** No camina sino que se oculta debajo de su casco de gran resistencia a los disparos. Al levantar su casco arroja 3 disparos en distintos ángulos para dañar a Megaman.

**Resistencia:** Oculto bajo su casco no puede ser dañado. Cuando no está oculto debe recibir 1 disparo para ser destruido.

**Resistencia a armas especiales:** Cuando está oculto bajo su casco sólo puede ser

	dañado por el arma de Bombman o Sparkman. Las otras armas especiales lo destruyen con un disparo.
 <b>Bumby</b>	<p><b>Descripción:</b> Vuela gracias a su hélice de gran poder. Arroja disparos desde lo alto que dañan a Megaman.</p> <p><b>Resistencia:</b> Aunque es difícil de alcanzar, es destruido con 1 disparo de plasma.</p> <p><b>Resistencia a armas especiales:</b> Cualquier arma especial lo destruye con 1 disparo.</p>
 <b>Sniper</b>	<p><b>Descripción:</b> Se oculta detrás de su escudo para protegerse. Para disparar con su rifle debe bajar el escudo quedando al vulnerable.</p> <p><b>Resistencia:</b> Detrás de su escudo no es alcanzado por los disparos de plasma. Bajo disparos directos soporta hasta 2 proyectiles antes de ser destruido.</p> <p><b>Resistencia a armas especiales:</b> Oculto bajo su escudo es destruido con las armas de Fireman o Ringman (2 disparos). En una posición vulnerable cualquier arma especial lo destruye con 1 disparo.</p>
 <b>Jumping Sniper</b>	<p><b>Descripción:</b> Al igual que Sniper, se oculta detrás de su escudo para protegerse pero en este caso, realiza saltos entre disparo y disparo.</p> <p><b>Resistencia:</b> Detrás de su escudo no es alcanzado por los disparos de plasma. Bajo disparos directos soporta hasta 2 proyectiles antes de ser destruido.</p> <p><b>Resistencia a armas especiales:</b> Oculto bajo su escudo es destruido con las armas de Fireman o Ringman (2 disparos). En una posición vulnerable cualquier arma</p>

## Obstáculos y Escenario

Los distintos niveles se conforman en base a un escenario de fondo y obstáculos dispuestos en distintos lugares. Los niveles, a su vez poseen puntos de generación de enemigos (*spawns*). Megaman puede avanzar y retroceder en el escenario pero, a medida que lo hace, los enemigos que había derrotado vuelven a aparecer. Los obstáculos pueden servirle de utilidad para alcanzar las zonas más elevadas ya que el salto de Megaman no es muy alto.

Existen 3 tipos de obstáculos:

- **Bloques:** Se los puede encontrar de 3 colores aunque no hay diferencia entre ellos.
- **Puas:** Son tan filosas que destruyen a Megaman al simple contacto.
- **Escaleras:** No ofrecen obstáculo a Megaman pero al toparse con una escalera puede utilizarla para subir o bajar a lugares poco accesibles utilizando los comandos de 'arriba' o 'abajo'.

Ninguno de los obstáculos puede ser destruido con los disparos de Megaman ni sus enemigos. En el caso de las escaleras, tanto los disparos como los personajes pueden atravesarlas sin problemas.

Como agregado, al caer en precipicios, Megaman es destruido. De esta forma es importante marcar las zonas de precipicio para poder detectar esta situación.

Al llegar al final del nivel, Megaman entra en una cámara especial donde se encuentra el jefe del nivel. Al ingresar no es posible salir de la cámara y debe combatir a muerte contra el enemigo de turno.

# Dinámica de Juego

## Partidas y Jugadores

No existen puntos ni hay tiempo para completar los niveles. El juego consta de partidas en las que colaboran hasta 4 jugadores con el objetivo de derrotar a los 5 robots para restaurar el orden.

Los distintos jugadores se registran en el servidor, reciben número y con ello un Megaman para iniciar juntos la partida.

Al ingresar el primer jugador se define una nueva partida donde los siguientes jugadores se unirán automáticamente previo ingreso de su nombre. Todos los jugadores que se unan a la partida recibirán notificaciones sobre los compañeros que ingresen pero sólo el primer jugador podrá iniciar el juego y elegir el próximo nivel. El límite de 4 jugadores será validado al iniciar la partida ya que, una vez iniciado el juego, no se admitirán nuevos jugadores.

Cada Megaman recibirá un total de 3 vidas al inicio que podrá aumentar al encontrar el *power-up* correspondiente. Cuando un jugador muere, los otros participantes continúan jugando en el nivel hasta finalizarlo o morir. Si todos los jugadores mueren, el nivel vuelve a iniciar salvo que hayan llegado a la cámara del jefe. En ese caso, todos los jugadores vuelven a dicha cámara para continuar desde allí.

Cuando las vidas de un jugador se acaban, no es regenerado en el próximo inicio de nivel y, en caso de que no quede ningún jugador con vidas, se declara el fin del juego reiniciando el ciclo de espera de participantes. Todo Megaman posee una barra de energía que disminuye al ser dañado por un enemigo. Cuando se agota la energía, Megaman muere perdiendo una vida.

## Poderes Especiales

Luego de derrotar a cada jefe, todos los jugadores obtienen un nuevo tipo de disparo en su cañón. Este nuevo poder tendrá características similares al que posee el jefe derrotado y podrá ser alternado con el cañón de plasma standard mediante los botones 1 al 5 (de acuerdo al orden en que sean vencidos los jefes). Las armas especiales no tienen municiones infinitas como en el caso del cañón de plasma. Al contrario, cada jugador que elija utilizar algún poder especial notará una nueva barra indicadora en su pantalla que bajará a medida que utilice el arma. Los poderes especiales pueden recargarse al encontrar cápsulas de plasma siempre que se tenga un arma especial seleccionada.

## Premios o *power-up*

Al destruir a los enemigos es posible obtener un *power-up* que mejora las habilidades del jugador que lo recoja:

<b>Nueva Vida</b>	<ul style="list-style-type: none"><li>• Aumenta en 1 la cantidad de vidas que posee el jugador. Si este tenía la cantidad máxima de vidas posibles, no tiene efecto.</li><li>• <b>Probabilidad de Aparición:</b> 0.01</li></ul>
<b>Cápsula de Energía Pequeña</b>	<ul style="list-style-type: none"><li>• Incrementa la cantidad de energía del jugador en 2 unidades hasta llegar al máximo .</li><li>• <b>Probabilidad de Aparición:</b> 0.1</li></ul>
<b>Cápsula de</b>	<ul style="list-style-type: none"><li>• Incrementa la cantidad de energía del jugador en 6 unidades hasta</li></ul>

<b>Energía Grande</b>	llegar al máximo. • <b>Probabilidad de Aparición:</b> 0.05
<b>Cápsula de Plasma Pequeña</b>	• Incrementa el poder especial del arma seleccionada en 2 unidades hasta llegar al máximo. • <b>Probabilidad de Aparición:</b> 0.1
<b>Cápsula de Plasma Grande</b>	• Incrementa el poder especial del arma seleccionada en 6 unidades hasta llegar al máximo. • <b>Probabilidad de Aparición:</b> 0.05

La probabilidad de que al eliminar un enemigo no se libere un *power-up* es  $1-P(\text{aparición de algún elemento})$  es decir  $1-(0.01+0.1+0.05+0.1+0.05) = 0.69$

## Aplicaciones Requeridas

El juego se dividirá en tres aplicaciones: cliente, servidor y editor de niveles. A continuación se describen estos sistemas.

### Servidor

Se trata de una aplicación de consola que permite a los Clientes conectarse entre sí y participar de una partida. Si bien no posee interfaz gráfica, es responsabilidad del servidor ejecutar la lógica del juego y notificar a los clientes acerca del estado de la partida para que estos informen al usuario.

El servidor posee el detalle de obstáculos y enemigos de cada nivel que será ofrecido a los jugadores al crear una partida. El contenido de los niveles sólo podrá ser modificado por administradores del servidor y requerirá reiniciar el sistema para que tenga efecto.

El servidor debe ser pensado para ejecuciones por períodos prolongados esperando que sea robusto en cuanto al manejo de recursos y notificaciones a los clientes conectados. Finaliza su ejecución al recibir un caracter 'q' por entrada estándar.

### Física y detección de colisiones:

La física del juego es muy básica pero importante para el correcto funcionamiento del juego. Se debe resolver el movimiento rectilíneo de los jugadores y disparos. De la misma forma, se debe resolver la aceleración durante los saltos y caídas. Por último, la detección de colisiones debe existir aunque su precisión puede ser ajustada si la velocidad con que se la ejecuta es suficiente para la velocidad de los elementos (evitando así que los objetos se 'traspasen').

A tal fin, se puede implementar el modelo físico simple con formas rectangulares o utilizar una librería como *box2d* [5].

### Logging

El servidor debe contar con un Log de eventos de fácil lectura. Ejemplos de estos eventos son:

- Cliente X conectado
- Cliente X desconectado
- Partida - Y jugadores unidos. Esperando confirmación de X para iniciar el juego.

- Partida - Jugador 1 - Disparo recibido. Energía restante: Z.

Para implementar este requerimiento no está permitido utilizar las salidas estándar o de error. Se debe usar un framework de logging reconocido en el mercado (log4cplus [1], glog [2] o log4cpp [3]) y configurar un archivo de salida para los mensajes.

## Configuración

La configuración de las variables del juego debe manejarse en un archivo de texto y no como información fija en el código fuente (*hardcodes*). A tal fin, se debe utilizar algún esquema de variables de configuración por XML o texto plano clave/valor que permita modificar:

- Velocidad de cada personaje (por tipo).
- Cantidad de vidas iniciales de cada jugador.
- Velocidad del juego.
- Velocidad máxima de disparos.
- Tiempo de recuperación que toma a los jugadores luego de un disparo.
- Etc.

## Cliente

Aplicativo de interfaz gráfica que se conecta con el servidor para permitir al jugador crear o unirse a la partida existente.

Sólo puede dar comienzo a la partida el primer jugador conectado y de esta manera se deja de aceptar nuevos jugadores.

Una vez iniciada la partida, el cliente debe mostrar el escenario en modo 'pantalla completa' (o similar de acuerdo con las posibilidades técnicas). Los jugadores no pueden avanzar independientemente en el nivel sino que lo deben hacer en conjunto. Cuando todos los jugadores avanzan más allá del centro de la pantalla, el área de visión se mueve con ellos. Si algún jugador queda rezagado, los otros jugadores no pueden continuar.

La aplicación cliente no debe ejecutar lógica del modelo del juego sino que debe estar pendiente de los eventos que ocurren en el servidor para graficar los cambios en el nivel. Asimismo, debe enviar al servidor las acciones de teclado recibidas por parte del usuario para que éste elabore una respuesta.

Se sugiere la utilización de *cairomm* [4]

## Imágenes

El juego debe contar con varias imágenes para cada personaje y alternarlas de acuerdo a su movimiento. Siendo que el jugador toma una visión parcial del escenario, es muy importante que las imágenes permitan identificar la dirección y sentido del movimiento en los personajes. Es importante tener en cuenta la posición de cada jugador y sus dimensiones para la verificación de choques. Si las imágenes difieren del área de choque disminuye el nivel de realismo de forma notoria.

## Editor de Niveles

Editor gráfico para el alta, modificación y eliminación de niveles. Permitirá editar la estructura física del nivel,

la ubicación de obstáculos y puntos de generación de jugadores o enemigos.

Mediante un *toolbox* con los distintos elementos del juego se debe permitir la selección y posicionamiento de los mismos dentro del escenario. Si bien la opción de *drag and drop* resulta ideal, es posible utilizar otros mecanismos gráficos para lograr el posicionamiento. De la misma manera, se debe permitir modificar la posición de los elementos ya ubicados en el escenario (por ejemplo, arrastrándolos y soltándolos en la nueva posición). Los enemigos serán ubicados mediante puntos de generación (o *spawns*) para cada tipo. Se debe indicar el punto de generación de los 4 posibles jugadores para garantizar el inicio de la partida así como el jefe del nivel y su cámara.

El editor debe contar con validaciones especiales para garantizar que cada nivel es consistente. Ejemplos de las validaciones necesarias son:

- Existencia de 4 generadores de jugadores.
- Existencia de un único jefe y la definición de su correspondiente cámara.
- Existencia de al menos un *spawn* de enemigos (de cualquier tipo).
- Evitar la superposición de elementos.
- Obstáculos correctamente definidos.

## Distribución de Tareas Propuesta

Con el objetivo de organizar el desarrollo de las tareas y distribuir la carga de trabajo, es necesario planificar las actividades y sus responsables durante la ejecución del proyecto. La siguiente tabla plantea una posible división de tareas de alto nivel que puede ser tomada como punto de partida para la planificación final del trabajo:

	Alumno 1 Servidor - Modelo	Alumno 2 Modelo - Cliente	Alumno 3 Cliente - Editor
<b>Semana 1</b> (3/5/2016)	<ul style="list-style-type: none"><li>- Definición del modelo de concurrencia y protocolo de comunicaciones</li><li>- Versión básica de sistema servidor</li><li>- Diagramas de clases</li></ul>	<ul style="list-style-type: none"><li>- Repositorio compartido y compilación automática (CMake, autotools o similar) p/3 ejecutables</li><li>- Definición de interfaz gráfica y flujos de uso</li><li>- Aplicación básica en <i>gtkmm</i> usando <i>cairomm</i></li></ul>	<ul style="list-style-type: none"><li>- Definición de interfaz gráfica y flujos de uso</li><li>- Aplicación básica en <i>gtkmm</i> usando Glade</li><li>- Diagramas de navegación (<i>wireframes</i>)</li></ul>
<b>Semana 2</b> (10/5/2016)	<ul style="list-style-type: none"><li>- Implementación del protocolo con servicios y resultados ficticios.</li><li>- Implementación básica del modelo de juego.</li><li>- Logging.</li></ul>	<ul style="list-style-type: none"><li>- Dibujos de <i>sprites</i> en Cairo/SDL</li><li>- Captura de teclado</li><li>- Movimiento veloz de <i>sprites</i> en base al teclado.</li></ul>	<ul style="list-style-type: none"><li>- Aplicación estática editora con Glade</li><li>- Lectura y escritura de archivo XML/JSON de niveles con datos ficticios.</li></ul>
<b>Semana 3</b> (17/5/2016)	<ul style="list-style-type: none"><li>- Modelo de partidas y niveles.</li><li>- Lectura de XML/JSON de niveles.</li><li>- Solución concurrente para aceptación de</li></ul>	<ul style="list-style-type: none"><li>- Comunicación con servicios del Servidor.</li><li>- Empleo de imágenes secuenciales para el movimiento de jugadores.</li></ul>	<ul style="list-style-type: none"><li>- Toolbox y creación de imágenes para reutilizar en cliente.</li><li>- Drag and Drop de elementos al nivel.</li><li>- Grabación básica de</li></ul>



	clientes y ejecución de juegos.		niveles
<b>Semana 4</b> (24/5/2016)	<ul style="list-style-type: none"> <li>- Mejoras en lógica de modelo.</li> <li>- Integración del protocolo con el modelo dinámico.</li> <li>- Configuración de parámetros basados en archivos de texto.</li> </ul>	<ul style="list-style-type: none"> <li>- Flujo de pantallas para Crear y Unirse a partidas.</li> <li>- Reproducción de Sonidos.</li> <li>- Configuración de parámetros basados en archivos de texto.</li> </ul>	<ul style="list-style-type: none"> <li>- Abrir, Guardar y Guardar Como finalizados.</li> <li>- Drag and Drop de elementos nuevos y existentes.</li> <li>- Altas, Bajas y Modificaciones de niveles finalizados.</li> </ul>
<b>Semana 5</b> (31/5/2016)	<ul style="list-style-type: none"> <li>- Pruebas y corrección sobre estabilidad.</li> <li>- Detalles finales y documentación preliminar</li> </ul>	<ul style="list-style-type: none"> <li>- Pruebas y correcciones en la jugabilidad.</li> <li>- Detalles finales y documentación preliminar</li> </ul>	<ul style="list-style-type: none"> <li>- Rotación y atributos extra para cada elemento.</li> <li>- Detalles finales y documentación preliminar</li> </ul>
<b>Preentrega (7/6/2015)</b>			
<b>Semana 6</b> (7/6/2016)	<ul style="list-style-type: none"> <li>- Testing y corrección de bugs</li> <li>- Documentación</li> </ul>	<ul style="list-style-type: none"> <li>- Testing y corrección de bugs</li> <li>- Documentación</li> </ul>	<ul style="list-style-type: none"> <li>- Testing y corrección de bugs</li> <li>- Documentación</li> </ul>
<b>Semana 7</b> (14/6/2016)	<ul style="list-style-type: none"> <li>- Correcciones sobre Preentrega</li> <li>- Testing</li> <li>- Documentación</li> <li>- Armado del entregable</li> </ul>	<ul style="list-style-type: none"> <li>- Correcciones sobre Preentrega</li> <li>- Testing</li> <li>- Documentación</li> <li>- Armado del entregable</li> </ul>	<ul style="list-style-type: none"> <li>- Correcciones sobre Preentrega</li> <li>- Testing</li> <li>- Documentación</li> <li>- Armado del entregable</li> </ul>
<b>Entrega (21/6/2016)</b>			

## Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema se debe realizar en ISO C++ utilizando librerías *gtkmm* y/o *SDL*[6].
2. Con el objetivo de facilitar el desarrollo de las interfaces de usuario, se permite el uso de *Glade*.
3. La información de niveles deben ser almacenados en formato XML o JSON. A tal fin, y con el objetivo de minimizar tiempos y posibles errores, se permiten distintas librerías externas (consultar sitio de la cátedra). No está permitido utilizar una implementación propia de lectura y escritura de XML o JSON.
4. Es condición necesaria para la aprobación del trabajo práctico la entrega de la documentación mínima exigida (consultar sitio de la cátedra). Es importante recordar que cualquier elemento faltante o de dudosa calidad pone en riesgo la aprobación del ejercicio.
5. De forma opcional, se sugiere la utilización de alguna librería del estilo *xUnit* [7]. Si bien existen varias librerías disponibles en lenguaje C++ [8], se recomienda optar por *CxxTest* [9] o *CppUnit* [10].

## Referencias

[1] Log4cplus: <http://sourceforge.net/p/log4cplus/wiki/Home/>

[2] Glog: <https://google-glog.googlecode.com/svn/trunk/doc/glog.html>

- [3] Log4CPP: <http://log4cpp.sourceforge.net/>
- [4] cairomm: <http://www.cairographics.org/cairomm/>
- [5] box2d: <http://box2d.org/>
- [6] SDL: <https://www.libsdl.org/>
- [7] Frameworks XUnit: <http://en.wikipedia.org/wiki/XUnit>
- [8] Variantes XUnit para C/C++: [http://en.wikipedia.org/wiki/List\\_of\\_unit\\_testing\\_frameworks#C.2B.2B](http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks#C.2B.2B)
- [9] CxxTest: <http://cxxtest.com>
- [10] CppUnit: [http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main\\_Page](http://sourceforge.net/apps/mediawiki/cppunit/index.php?title=Main_Page)