**Documentación técnica**

**Ingeniería de Software**

*Buscaminas-Singletonsgroup*

*IceScrum:[https://lsi.vc.ehu.eus/icescrum/login?ref=p/PROJ19/#feature](https://lsi.vc.ehu.eus/icescrum/login?ref=p/PROJ19/" \l "feature)*

*Github:*[*https://github.com/EHUGasteiz/buscaminas-singletonsgroup*](https://github.com/EHUGasteiz/buscaminas-singletonsgroup)

*Sergio Vergara*

1. **Introducción**

El proyecto buscaminas de la asignatura Ingeniería de Software consiste en el diseño e implementación de una aplicación basada en el popular juego del Buscaminas conocido como uno de los juegos del Sistema Operativo Microsoft Windows. La lógica de juego se fundamenta en descubrir casillas de un panel de forma que cada casilla tenga las opciones de, casilla con mina por lo que se marca con una bandera, y casilla sin mina con lo que se descubre señalizando con un número representando las minas adyacentes. El juego concluye cuando una mina no es marcada con la bandera y se descubre, y cuando se han señalizado todas la minas en el panel y descubierto también las casillas sin mina. El juego finalmente permite grabar al ganador su nombre y una puntuación, que se muestra en una tabla con el resto de ganadores.

Los conceptos de programación e Ingenería de Software a trabajar en el proyecto son los siguientes:

- Utilización de estándares de Java 8 (streams, collectors, etc) y trabajo de datos con Java-XML

- Utilización de Java swing para la construcción del entorno gráfico basado en JPanel, JButton, JLabel, JMenu...y su manejo basado en Layouts, GridBagConstraints, etc

- Utilización del estilo de arquitectura de Software Modelo Vista Controlador, mediante el cual se separa el código referente al entorno gráfico y representación de la información, de la lógica de proceso de funcionalidades, y de interpretación de las mismas.

- Utilización de UML y patrones de diseño, de tal forma que el diseño de esta aplicación parte de una estructura preconcebida y bien representada mediante diagramas de clases y diagramas de secuencia, que permiten proyectar una idea de programación y de lógica funcional previamente a la fase de desarrollo e implementación.

- Utilización de la librería Junit para la realización de tests.

1. **Organización**

El trabajo se ha desarrollado teniendo en cuenta los diferentes roles existentes en el entorno de la Ingeniería de Software, programación...como son: Jefes de Programación, Jefe de Proyecto, Testers, Analistas, Programadores, Diseñadores, y sus respectivas subcategorizaciones Junior, Senior, etc.

Es importante destacar, que este proyecto ha estado compuesto por dos integrantes sólamente, por lo que la acumulación de tareas en sólo nosotros dos ha limitado en cierto modo poder cumplir con exactitud los objetivos. Es por esto que las tareas de diseño, de control de proyecto mediante herramientas de Github y sobre todo programación, han ocupado la mayor parte de nuestro tiempo invertido en el proyecto.

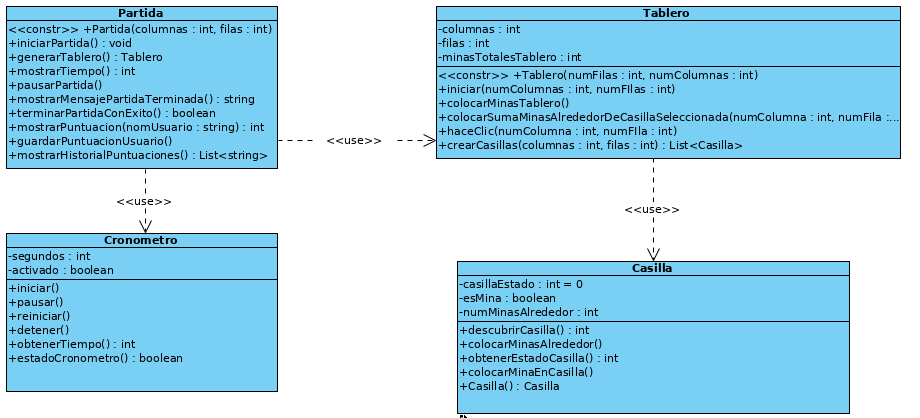
Hemos utilizado en la medida que nos ha sido posible Icescrum, de forma que hemos ido situando el trabajo en los estados “Hecho”, “En proceso” y “Finalizado”. La herramienta fundamental ha sido Eclipse, que nos ha permitido programar con cierta agilidad. También Visual Paradigm nos ha permitido crear una idea inicial en UML de lo que luego sería el programa Buscaminas. Github ha sido también importante, ya que nos permitido concentrar los desarrollos que individualmente íbamos haciendo en nuestro ordenador particular. También queremos remarcar que Github también suponía las menores veces un bloqueo, al tener que realizar y analizar “merges” con códigos añadidos al repositorio al mismo tiempo por parte de cada uno de los miembros del grupo. Es por esto que la práctica de desarrollo por turnos en ciertos puntos del proyecto nos ha facilitado el proceso.

El transcurso del proyecto se puede resumir en la idea común de los componentes del grupo de lo que sería el producto final, planteando al comienzo de la semana los objetivos a conseguir. Estos, estructurados y planificados en Historias de Usuario, se irían cumpliendo según la programación fuese más sencilla o más compleja en cada momento, y según las orientaciones por parte del profesor en las reuniones SCRUM del grupo con él. Las reuniones entre los integrantes del grupo han sido todas no presenciales, utilizando indistintamente conversaciones telefónicas, email, mensajería instantánea, Issues de Github, y compartiendo pantalla por videconferencia.

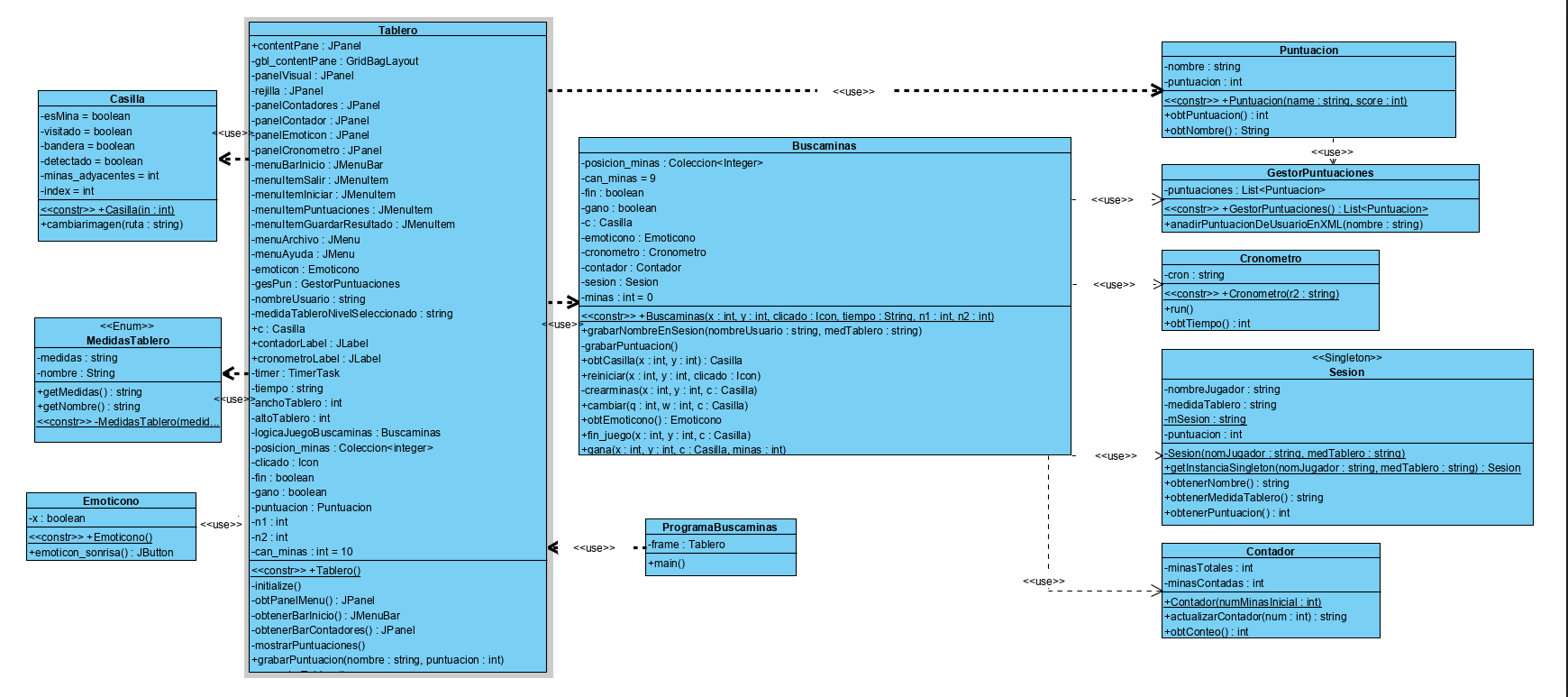
<https://github.com/EHUGasteiz/buscaminas-singletonsgroup/issues>

1. **Diseño completo**

Diseño inicial, diagrama de clases (Justo cuando se anunció la temática del proyecto. La teoría sobre patrones apenas había empezado) Utilizamos este diseño para partir de una idea inicial común ambos programadores. A medida que transcurría el curso, iba siendo adaptado a los conocimientos que ibamos adquiriendo:

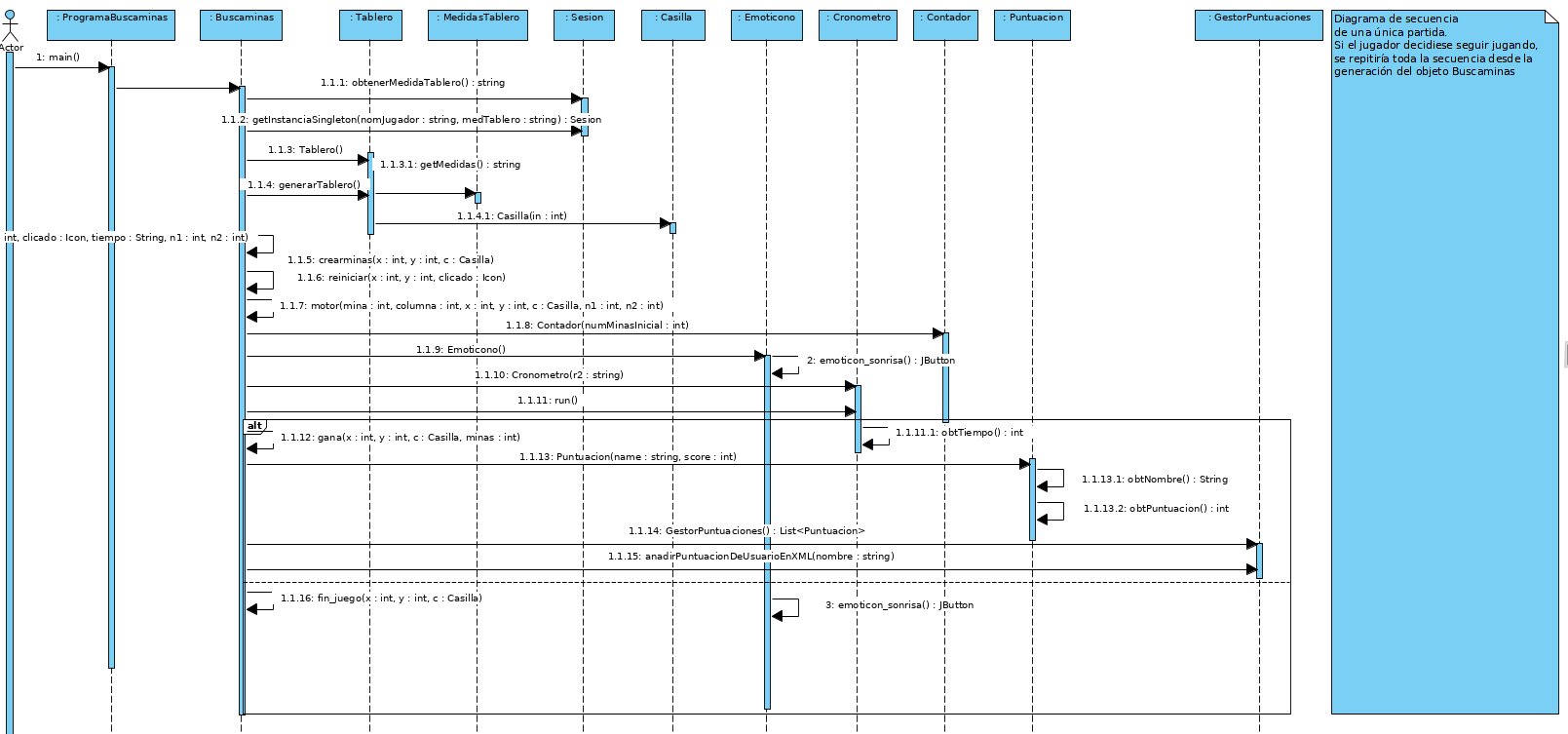


Diseño UML final, diagrama de clases:



Razonamiento: Hemos situado a la izquierda, las clases que se localizarían en la vista, relacionadas con la construcción de componentes gráficos. En el centro las clases que componen el modelo, encargadas de la lógica funcional del programa. La clase ProgramaBuscaminas la hemos situado en el centro por no tener espacio libre a la derecha del todo, donde situariamos esta clase localizada en el controlador.

Diagrama de secuencia:



Razonamiento del diagrama de secuencia: Se genera el Frame contenedor. Luego los paneles del menú de opciones, contador, emoticono, cronómetro. El usuario elige un nivel de dificultad y medida de tablero, se generan las casillas y el motor de juego. Se inicia la sesión. Transcurre el juego, si gana, el nombre de jugador y una puntuación se guardan, y si pierde el emoticono cambia y se muestran todas las casillas descubiertas

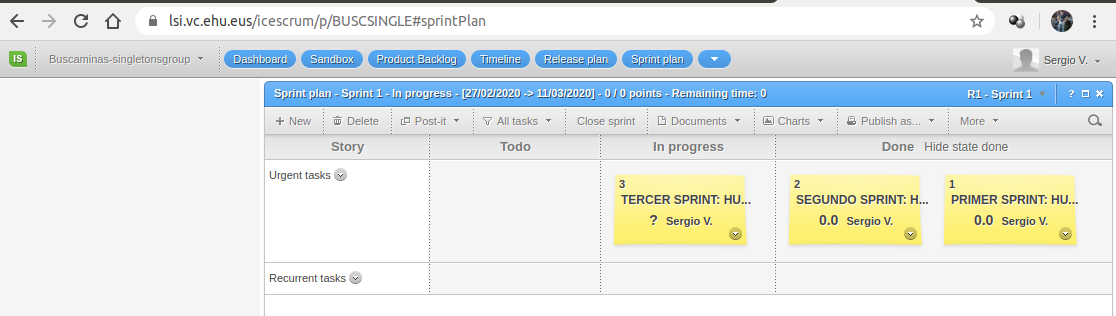
1. **Desarrollo**

Objetivos para cada Sprint en forma de Historias de Usuario proporcionados por el profesor:

- PRIMER SPRINT: HU2, HU3

- SEGUNDO SPRINT: HU1, HU4

- TERCER SPRINT: HU5, HU6, HU7



Composición de cada Historia de Usuario:

- HU1: Iniciar sesión con un nivel de dificultad

- HU2: Generar Buscaminas del nivel seleccionado

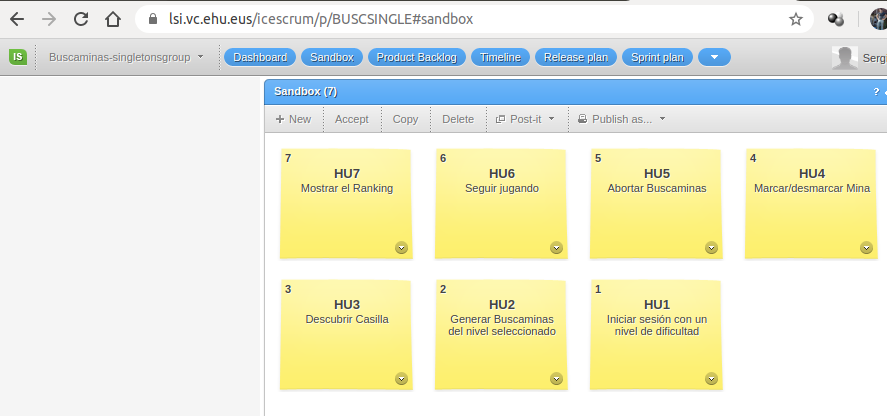
- HU3: Descubrir Casilla

- HU4: Marcar/desmarcar Mina

- HU5: Abortar Buscaminas

- HU6: Seguir jugando

- HU7: Mostrar el Ranking



A la hora de implementar las distintas Historias de Usuario, los programadores detectábamos más factible adelantar desarrollos de próximos SPRINTS por adecuación del código ya presente y mayor facilidad de ese momemnto y, por tanto, ahorro de tiempo, y por una más correcta distribución de tareas entre programadores. Por esto, se comunicaban estas facilidades al Jefe de Proyecto, el cual adecuaba el transcurso del proyecto, trabajo con Issues de Github y reuniones con el cliente en favor del beneficio global del proyecto.

1. **Conclusiones**

La reflexión del proyecto ha sido muy positiva, ya que hemos tenido la oportunidad de practicar diversos conceptos del mundo de la programación e Ingeniería de Software con los estándares más actuales, tecnología más moderna, entornos óptimos de programación y mejorar nuestras destrezas con el tan utilizado lenguaje de programación Java.

En caso de repetirlo, se cambiarían la forma de desarrollo de la aplicación, dando aún más relevancia a la fase de diseño que pueda ahorrar tiempo de programación, y la estructuración Modelo Vista Controlador, haciendo que las diversas capas estuviesen aún más diferenciadas y desacopladas. También utilizaríamos en mayor medida el patrón Singleton, como por ejemplo en Contador, Cronometro y en la propia clase Buscaminas. Por esto puede ser interesante reincidir en el tiempo limitado para este desarrollo, que ha su vez no sólo implicaba el desarrollo de una apliación nueva para los integrantes del grupo, sino también de la inclusión de nuevos estándares de Java, nuevos patrones de diseño, utilización de UML, sistema de desarrollo colaborativo mediante git (con sus beneficios en el desarrollo, pero también con las pérdidas de tiempo derivadas de ciertas operaciones “merge” y a veces necesidad de hacer reset, revert,etc). También, en caso de repetirlo, haríamos un código mejor estructurado, con unos nombres de clases y métodos mucho más precisos respecto a su funcionalidad, y con la experiencia de haberlo hecho ya una vez, seguramente tendriamos más tiempo para hacer un frontend responsive. Y para finalizar, cambiaríamos el número de integrantes del grupo, ya que la inversión de tiempo de los sólamente dos recursos de este proyecto, ha sido considerablemente grande y de alguna forma extraordinario.

1. **Anexos**

Documentación de tareas:

- <https://github.com/EHUGasteiz/buscaminas-singletonsgroup/issues>

- <https://github.com/EHUGasteiz/buscaminas-singletonsgroup/commits/master>

Documentación de las pruebas: A continuación mostramos en una tabla los test que hemos creado más importantes, ya que entendemos que una mejor forma de ver los test es ejecutándolos desde Eclipse y comprobando que ciertamente pasan sin errores. También hemos realizado varias partidas manualmente simulando diversas aciones del usuario. En esta última situación, al detectar errores y parchearlos, indicábamos ún único número de Issue en Github creado para recoger los bugs detectados en las ruebas manuales.

Tabla con los principales casos de prueba:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id. de caso de prueba** | **Objetivo** | **Entrada** | **Condiciones de ejecución** | **Resultado esperado** | **Resultado obtenido** | **Comentario** |
| **ProgramaBuscaminasTest.java** | Comprobar objeto no es null | Assert.assertNotNull(frame); | Tablero frame = new Tablero(); | Pasa | Pasa | Todo bien |
| **ProgramaBuscaminasTest.java** | Comprobar objeto no es null | Assert.assertTrue(resultVisible); | boolean resultVisible = frame.isVisible(); | Pasa | Pasa | Todo bien |
| **ContadorTest.java** | Comrpobar el contador de minas no contiene valores negativos | Assert.assertNotEquals(numMinasInicial, -1);  Assert.assertNotEquals(numMinasInicial, -2);  Assert.assertNotEquals(numMinasInicial, -3);  Assert.assertTrue(numMinasInicial>-1); | ct = new Contador(minasTotales);  int numMinasInicial = ct.obtConteo(); | Pasa | Pasa | Todo bien |
| **ContadorTest.java** | Comprobar el contador suma de uno en uno | Assert.assertNotEquals(ct.obtConteo(), -2);  Assert.assertNotEquals(ct.obtConteo(), 2);  Assert.assertNotEquals(ct.obtConteo(), 0); | ct = new Contador(9);  ct.actualizarContador(1); | Pasa | Pasa | Todo bien |
| **ContadorTest.java** | Comprobar que obtConteo tiene valor inicial cero, y cuenta correctamente | Assert.assertEquals(ct.obtConteo(), i); | ct =new Contador(9);  int i=1;  ct.actualizarContador(i); | Pasa | Pasa | Todo bien |
| **ContadorTest.java** | Comprobar funciona el contador incrementando | (Entradas se intercalan con los asserts, por lo que están en la siguiente celda) | ct = new Contador(9);  Assert.assertEquals(ct.obtConteo(), 0); ct.actualizarContador(1);  Assert.assertEquals(ct.obtConteo(), 1); | Pasa | Pasa | Todo bien |
| **ContadorTest.java** | Comprobar funciona el contador decrementando | ct = new Contador(9);ct.actualizarContador(1);  ct.actualizarContador(-1); | Assert.assertEquals(ct.obtConteo(), 0); | Pasa | Pasa | Todo bien |
| **ParameterizedTestContadorUsingConstructor.java** | est constructor Contador.java | Contador tester = new Contador(); | assertEquals("Result", m1 \* m2, tester.multiply(m1, m2)); | Pasa | Pasa | Todo bien |
| **CronometroTest.java** | Comprobar que el valor inicial del cronómetro es 0 | CronometroTest cronometroTest = new CronometroTest("0"); | Assert.assertEquals(cronometroTest.obtTiempoTest(), 0); | Pasa | Pasa | Todo bien |
| **CronometroTest.java** | Comprobar que el cronometro funciona | CronometroTest cronometroTest = new CronometroTest("0"); | cronometroTest.run(); | Pasa | Inicia, pero Junit no es capaz de detenerlo y pausa el resto de test | Test pausado en el thread |
| **SesionTest.java** | Comprobar que la sesion se crea correctamente | Sesion sesionTest= Sesion.getInstanciaSingleton(nombreJugador, medidaTablero); | Sesion sesionTest= Sesion.getInstanciaSingleton(nombreJugador, medidaTablero); | Pasa | Pasa | Todo bien |
| **SesionTest.java** | Comprobar que la puntuacion obtenida es la que se ha establecido | Sesion sesionTest= Sesion.getInstanciaSingleton(nombreJugador, medidaTablero);  sesionTest.establecerPuntuacion(10); | Assert.assertEquals(sesionTest.obtenerPuntuacion(),10); | Pasa | Pasa | Todo bien |
| **CasillaTest.java** | Comprobar que el objeto de la clase Casilla inicia, e inicia en número variable de instancias | @Parameters  public static Collection<Object[]> data() {  return Arrays.asList(new Object[][] {  { 0 }, { 1}, { 20}, { 40 },{ 70 }, { 100}, { 150}, { 200}, { 1000}  });  }  @Parameter // primer valor es cero  public int entrada;    @Before  public void initialize() {  casilla = new Casilla(entrada);  casillaTest();  } | @Test  public void casillaTest() {  //Comprobar que la casilla inicialmente no es mina  Assert.assertFalse(casilla.esMina);  //Comprobar que la casilla inicialmente no tiene casillas adyacentes  Assert.assertEquals(casilla.minas\_adyacentes, 0);  //Comprobar que la casilla inicialmente no tiene valor visitado=true  Assert.assertFalse(casilla.visitado);  //Comprobar que la casilla inicialmente no tiene bandera=false  Assert.assertFalse(casilla.bandera);  //Comprobar que la casilla inicialmente no tiene valor detectado=false  Assert.assertFalse(casilla.detectado);  //Comprobar que la casilla inicialmente tiene tamano 25x25  Assert.assertEquals(casilla.getPreferredSize(),new Dimension(25,25));  } | Pasa | Pasa | Todo bien |
| **BuscaminasTest.java** | //Comprobar que los objetos relacionados con Buscaminas se inicializan sin errores y que el propio objeto Buscaminas se crea con las medidas preestablecidas | Buscaminas buscaminas;  Icon clicado;  @Parameters  public static Collection<Object[]> data() {  return Arrays.asList(new Object[][] {  { 7,10 }, { 10,15}, { 12,15}  });  }    @Parameter // primer valor es cero  public int entradaX, entradaY; | Emoticono emoticono = new Emoticono();  Cronometro cronometro = new Cronometro("0");  Contador contador = new Contador(9);  Object sesion = null;  clicado= new ImageIcon() ;  Puntuacion puntuacion = null; | Pasa | Pasa | Si |
| **BuscaminasTest.java** | //Comprobar que el objeto buscaminas se crea correctamente | Buscaminas buscaminas;  Icon clicado;  @Parameters  public static Collection<Object[]> data() {  return Arrays.asList(new Object[][] {  { 7,10 }, { 10,15}, { 12,15}  });  }    @Parameter // primer valor es cero  public int entradaX, entradaY; | buscaminas = new Buscaminas(entradaX, entradaY, clicado,"0", n1, n2); | Pasa | Pasa | si |
| **BuscaminasTest.java** | //Simular clics y que los estados de las casillas funcionan correctamente | buscaminas = new Buscaminas(12, 15, clicado,"0", n1, n2);  int numFilas = 11;  int numColumnas = 14;  Casilla c; | for (int i = 0; i < numFilas; i++) {  for (int j = 0; j < numColumnas; j++) {  //La colocación de las minas ya está hecha en este punto, luego no tiene sentido testear la variable c.esMina a true o false  c=buscaminas.obtCasilla(numFilas, numColumnas);  c.doClick();  c.detectado=true;  c.detectado=false;  c.bandera=true;  c.bandera=false;  c.visitado=false;  c.visitado=true;    }  } | Pasa | Pasa | si |