Tarea 3 - Breakout

Profesor: Alexandre Bergel
Auxiliar: Juan-Pablo Silva

Breakout

Breakout es un juego de *arcade* que consiste en utilizar una barra horizontal y una bola que rebota en las paredes de la pantalla, para golpear ladrillos posicionados en la parte superior de la pantalla. Cada vez que la bola choca con un ladrillo, esta rebota y puede destruirlo, atribuyendo un puntaje al jugador acorde a la dificultad de destruir dicho ladrillo. Si la bola toca la parte inferior de la pantalla, esta se considera una bola perdida y el jugador pierde una vida. Durante esta tarea 3 terminaremos de implementar este juego.

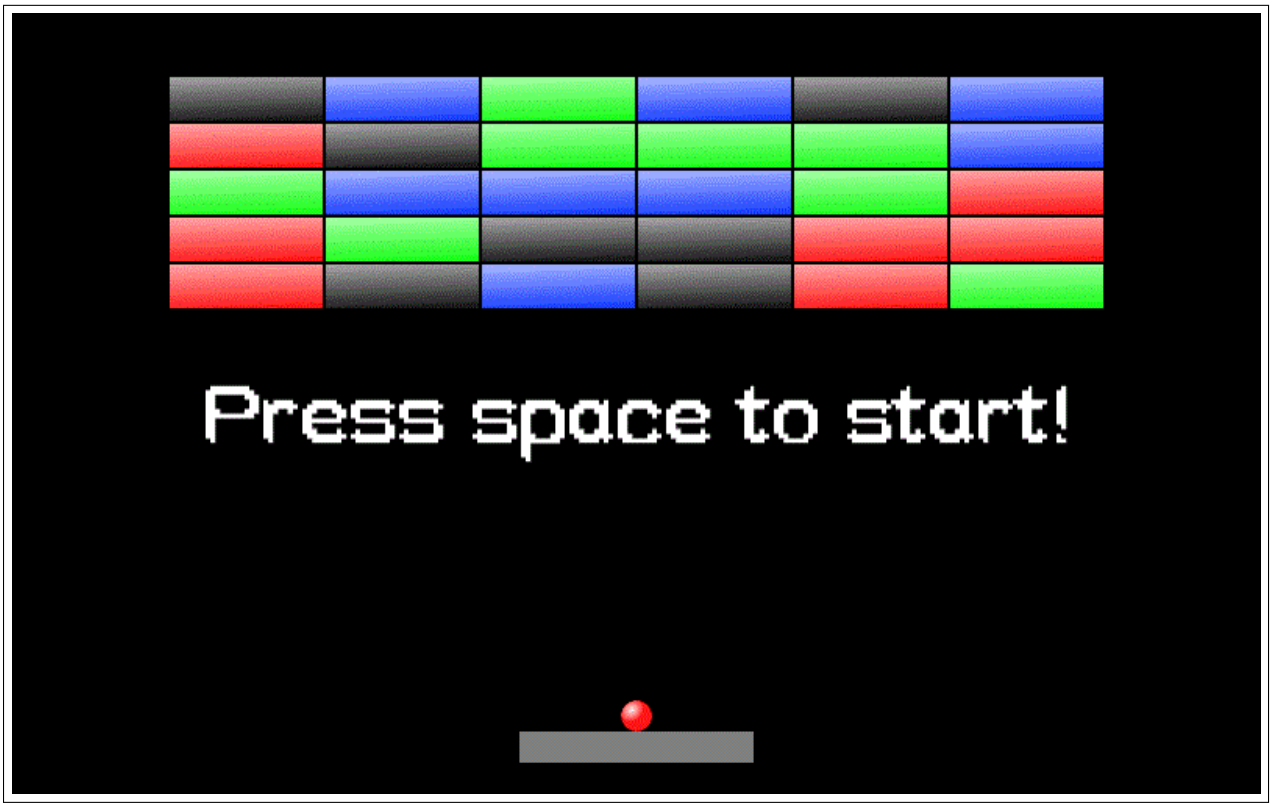

Figura 1: Ejemplo de un juego de Breakout.

El objetivo básico de *Breakout* es usar la bola para golpear *Brick*s y ganar puntos para pasar

al siguiente nivel. Los *Brick*s tienen la característica de que cuando la bola choca con ellos, esta rebota y, dependiendo del tipo de *Brick*, este puede ser destruido, y el jugador ganar puntos de ello. Una vez se haya obtenido todo el puntaje de un nivel, el juego continúa al nivel siguiente de forma automática. Cuando un jugador pierde todas las bolitas, este pierde el juego.

Requisitos

En esta tarea 3 usted deberá utilizar el código que implementó para la tarea 2, y agregar una interfaz gráfica al juego. Luego de implementar la interfaz gráfica, se le pedirá conectar la interfaz con la lógica del juego e implementar algunas de las *features* que se le presentarán en la sección *Features*. Además de esto, usted debe volver a su código de la tarea 2 y reflexionar sobre si, ahora que conoce más patrones de diseño, este provee flexibilidad y claridad a su código, o si necesita hacer *refactoring* y cambiar algunas cosas.

En esta ocación particular, se le entregará un código funcional del cual puede partir en caso de no haber terminado su tarea 2. En caso de utilizarlo debe indicar en su `readme` que así lo hizo. Se premiará a los estudiantes que utilizen su propio código para esta iteración, pero este debe estar completamente funcional y con un buen diseño. Se sugiere que también se indique si utilizaron su propio código.

Adicionales a Tarea 2

Deberá agregar al código de su tarea 2 las siguientes indicaciones:

- **Number Of Bricks:** el método *numberOfBricks* debe retornar el número total de *Brick*s que se encuentran actualmente sin destruir en el juego, y disminuir a medida que se destruyen *Brick*s; a diferencia de la tarea 2 donde no se consideraban los *MetalBrick*s. Esto aplica para el método en el `facade` y en *Level*.
- **Condición para pasar de nivel:** por problemas de enunciado se pudo haber entendido que la condición para pasar un nivel era destruir todos los *Brick*s del nivel. La condición correcta corresponde a obtener el puntaje máximo permitido por el nivel en cuestión.
- **Ayudas:** puede agregar los métodos *isWoodenBrick()*, *isGlassBrick()* y *isMetalBrick()* a la interfaz *Brick*, por si le son útiles para facilitar la desambiguación de tipos.

Elementos en pantalla

Existe un mínimo de elementos en pantalla que deben existir para considerar su interfaz gráfica como válida. Estas son:

- **Barra:** debe haber una barrita en pantalla. Lo común es que esta barra se encuentre en la parte inferior de la pantalla y solo pueda moverse horizontalmente. Es suficiente con que sea un rectángulo.
- **Bolas:** debe haber solo 1 bola en juego en todo momento. Las excepciones son: cuando la bola se pierde, que en ese momento no hay bola en juego, o que usted haya introducido un *Bonus* que permita que hayan más de 1 bola (esto es parte de los features opcionales más adelante). Las bolas deben ser *circulares*.
- **Bricks:** corresponden a los bloques dentro de un nivel y deben ser representados como rectángulos. Cada tipo *Brick* debe diferenciarse de los demás, es decir *GlassBrick*, *WoodenBrick* y *MetalBrick* deben tener texturas, colores, o algo que los distinga. En términos de posicionamiento, los bricks deben orientarse en filas uno al lado del otro, similar a como se muestra en la figura 1.
- **Información:** debe incluir información sobre bolas disponibles, número de niveles jugados y por jugar, puntaje actual y puntaje acumulado entre niveles, que se deben actualizar automáticamente cuando hay un cambio en alguno de estos valores. Actualmente en el código de su tarea 2 esto ya ocurre, pero debe mostrarlo dinámicamente en la interfaz gráfica tal que el jugador pueda ver sus avances.

Acciones mínimas

Para jugar *Breakout Pre-Alpha* se requieren interacciones entre el usuario (jugador) y el juego. Las acciones mínimas que deben implementarse pueden ser asignadas a las teclas que usted desee y corresponden a las siguientes:

- **Mover barra:** la barra debe moverse en 2 direcciones de forma horizontal. Usted debe permitir que al mantener presionada la tecla la *barra* se mantenga en movimiento, y que al soltarla la *barra* se detenga. Se sugiere usar la tecla `A` del teclado para moverse hacia una dirección, y la techa `D` para la dirección contraria, pero es libre de modificarlas.
- **Nueva bola:** el juego automáticamente crea una nueva bola en el punto medio de la barra, haciendo que esta no pueda moverse mientras no se ejecute la acción que se explicará a continuación. Esto es cierto para el inicio del juego y para cuando se pierde una bola cuando cae a la muralla inferior. Una nueva bola puede ser creada solo si quedan bolas disponibles, y en caso de que ya exista una bola en juego, no se puede crear una nueva hasta que se pierda. La acción consiste en que al presionar la tecla, la bola sale "disparada" desde la barra en alguna dirección. Se sugiere la tecla `ESPACIO` del teclado, pero es libre de cambiarla.
- **Nuevo nivel:** el juego se inicia sin un nivel configurado, por lo tanto no hay ningún *Brick* en la pantalla. Esta acción settea un nuevo nivel para el juego con parámetros por defecto que usted defina, pero debe ser un nivel completo, con todo tipo de *Brick*s. Se sugiere utilizar la

tecla `N`. Cada vez que se presione esta tecla se asignará un nuevo nivel, generado nuevamente usando los parámetros dados, agregando un nivel nuevo a la lista de niveles.

Interacciones en pantalla

En la pantalla ocurrirán interacciones entre objetos o cuando se presione un botón, estas son las consideraciones que debe respetar:

- **Barra:** la barra debe poseer un movimiento en una dimensión, y la bola rebotar al chocar con esta. La barra no puede salir de la pantalla.
- **Niveles:** al asignar un nuevo nivel al juego, los *Brick*s deben ser posicionados en filas de 10 *Brick*s cada una, dejando un espacio entre la fila de más arriba y el "techo" de la pantalla. Además, la lista de *Brick*s debe ser desordenada antes de mostrarlos en pantalla, de forma que no queden grupos de *Wooden* y *Glass* y al final todos los *Metal*. No debe preocuparse de casos en los que hayan demasiados *Brick* que las filas lleguen a la parte inferior de la pantalla.
- **Murallas:** la bola no debe salir del espacio visible de la pantalla, y debe rebotar en las murallas.
- **Rebotes:** la bola debe rebotar al golpear *Brick*s, opcionalmente puede implementar que los *Brick*s agreguen un impulso a la bola cuando esta los golpee.
- **Disparo de bola:** cuando se invoca una nueva bola, esta debe posicionarse en el centro de la barra, independiente de dónde esta se encuentre en al momento de hacer esta invocación. Aquí la barra se detiene esperando que la nueva bola sea "disparada" en alguna dirección.
- **Perder bola:** cuando una bola cae sobre la muralla inferior de la pantalla, se considera que la bola fue perdida y se pierde una "bola disponible". Cuando esto ocurre, además de perder una de las bolas, esta debe desaparecer de la pantalla y ser retirada del juego, para luego posicionar una nueva en el centro de la barra. Cláramente esto ocurre solo en caso de aún tener bolas disponibles.
- **Perder o ganar el juego:** cuando se pierda el juego (se perdieron todas las bolas disponibles) el jugador no puede seguir interactuando con el nivel y se le debe mostrar un mensaje de si quiere intentarlo de nuevo, y comenzar nuevamente el juego. Cuando el jugador gane, se le debe mostrar un mensaje de felicitaciones para luego darle la opción de volver a jugar.
- **Ayuda nuevo nivel:** debe implementar que al avanzar de nivel, los elementos del nivel anterior deben ser eliminados del juego antes de posicionar los elementos del nivel siguiente.

Features

Las secciones anteriores han presentado la funcionalidad *core* o base requerida para la tarea. A continuación se le presentan distintas descripciones de *features* que puede elegir implementar. Debe seleccionar mínimo 2 mayores y 2 menores, o la especial y 1 menor.

Especial

1. **Bonus:** añadir a los modelos del juego un nuevo conjunto de objetos que representen *Bonus*. Estos *Bonus* se activan bajo ciertas condiciones, como un `P` % de que se active un *Bonus* cuando un *Brick* sea destruido. Debe implementar al menos 2 *Bonus* que generen algo interesante dentro del juego, por ejemplo:

   - ***Bonus* de puntaje extra:** existe un *Bonus* que se activa con 3 % de probabilidad al destruir un *Brick*, que entrega 50,000 puntos, que deben ser actualizados automáticamente al puntaje acumulado del jugador. Esto no interfiere en ganar el nivel actual, solo se ve reflejado en el puntaje acumulado del jugador.
   - ***Bonus* de bola extra en juego:** existe un *Bonus* que al activarse invoca una nueva bola en la pantalla. Este *Bonus* se activa con un 5 % de probabilidad. Esto permite ganar el nivel más rápido, pero es más complicado mantener las bolas activas. Al haber más de una bola en juego, perder una bola no disminuye las bolas disponibles. Solo disminuye en una cuando se pierde la última bola en pantalla.
   - Usted puede proponer nuevos *Bonus*, pero deben tener algún tipo de dificultad. Preguntar al auxiliar.

   Además, debe existir algún tipo de señal o indicador visual de que un *Bonus* ha sido activado.

Mayores

1. **Estado distinto:** cuando un *Brick* es golpeado, este debe modificar cómo se visualiza, su color, o textura debe cambiar. Esto no es más que una indicación visual de que el *Brick* cambió de estado. En el caso de los *GlassBrick* esto no aplica ya que son destruidos instantáneamente, los *WoodenBrick* pueden mostrar daño al primer o segundo golpe, y los *MetalBrick* podrían modificarse 2 veces, la primera a los 3 golpes y la segunda a los 7, por ejemplo.
2. **Nuevo nivel configurable:** dentro de la aplicación, en una extensión de la interfaz para visualizar información del juego, debe existir un panel o menú que permita ingresar los parámetros deseados para crear un nuevo nivel y luego añadirlo. Estos parámetros serían el número de *Brick*s, la probabilidad de que un *Brick* sea un *GlassBrick*, la probabilidad de ocurrencia de un *MetalBrick* y el nombre del nivel. Luego con estos valores ingresados por el

usuario, se clickea un botón "Crear" para agregar el nuevo nivel a la lista de niveles en juego. El objetivo de esta funcionalidad es tener un nuevo nivel de forma más personalizada, ya que actualmente la acción de "nuevo nivel" es crear uno con parámetros aleatórios.

3. **Mecanismo de testing:** consiste en implementar alguna forma de testear manualmente el funcionamiento de las interacciones en la interfaz gráfica, por ejemplo permitiendo que un click sobre uno de los *Brick*s se considere un golpe, o poder mover la bola con el mouse, o alguna otra manera que a usted se le ocurra, permitiendo manualmente verificar que las interacciones efectivamente funcionen.

4. **Nuevo *Brick*:** usted debe implementar un nuevo tipo de *Brick* que tenga algún tipo de funcionalidad especial, como el *MetalBrick*. Esto quiere decir que no basta con que su distinción con el resto de los *Brick*s sea tener un puntaje y número de golpes requerido para ser destruido distinto. Este *Brick* debe tener un color o textura distinto al resto de forma que pueda identificarse.

Menores

1. **Sonido al golpe:** cuando la bola golpee un *Brick*, se debe reproducir un sonido que represente el golpe de la bola con la entidad. Se deben reproducir sonidos distintos para cada tipo de *Brick*, y además, debe haber un sonido especial para cuando la bola destruya un *Brick*, este sonido puede ser el mismo para todos.

2. **Bolas con imagenes:** en el sección de información de la interfaz gráfica se muestra el número de bolas disponibles para jugar. Esta opción consiste en reemplazar ese número por imágenes que representen cuántas bolas hay disponibles. Por ejemplo, mostrar 3 imagenes circulares de una bolita de goma cuando hay 3 bolas disponibles. Si el número de bolas aumenta o disminuye, el número de copias de la imagen debe variar como corresponda.

3. **Chispas/estrellas al golpear:** al golpear un *Brick* aparecen estrellas o chispas, **visualmente** aumentando el dramatismo del golpe. Esto se puede realizar animando particulas aleatórias que desaparezcan rápidamente para que no interrumpan el juego.

Jugabilidad

En esta tarea 3 usted tiene que conectar la lógica que implemetó para la tarea 2 con una interfaz gráfica. La interfaz gráfica debe mostrar las acciones automáticas que ocurren, como avanzar de nivel al completar el puntaje, perder el juego o ganar el juego, con algún indicador visual que muestre qué es lo que está ocurriendo.

Para comenzar a jugar se deben agregar niveles, ya sabemos que existe una acción con una tecla del teclado para activarla. Este nivel se añade al juego tal y como lo hacíamos en la ta-

rea 2, solo que ahora usted puede dejar los parámetros de probabilidad y número de *Brick*s como valores aleatórios. Respecto al *seed*, lo más simple es que usted use la siguiente llamada `System.currentTimeMillis()`, o si lo prefiere, puede crear otro método en el `facade` que no reciba un *seed* y llame al método antiguo con la llamada a `currentTimeMillis`.

Al ganar un nivel se deben borrar los elementos del nivel recién ganado para luego renderizar los elementos del nivel actual. Como aclaración, las bolas y puntaje acumulado por el jugador no dependen de un nivel específico, ya que se mantienen de nivel en nivel. El puntaje del nivel y su nombre depende del nivel actual.

Importante - Código base

En caso de usar el código base usted no puede copiar, modificar ni redistribuir ninguna parte del código. Se da expreso permiso de copiar y modificar el código para el uso particular de esta tarea 3, para uso privado no comercial. Usted puede distribuir su trabajo modificado, siendo este su tarea 3 final. Cualquier otro uso o incumplimiento de las condiciones se verá reflejado en un violación a los términos y condiciones del uso del software. Respete las condiciones de uso.

Notas adicionales

La implementación de la interfaz gráfica y controlador gráfico del juego deben hacerse en el paquete `gui`. Esta tarea debe ser realizada usando *JavaFX*. Se recomienda fuertemente usar la librería *FXGL*[1], que ya cuenta con un motor físico, de colisiones, gráfico y de inputs de usuario. La documentación de esta librería la puede encontrar en el siguiente enlace `https://github.com/AlmasB/FXGL/wiki`. Puede encontrar ejemplos de juegos implementados con esta libreria en `https://github.com/AlmasB/FXGLGames`. Para agregar esta librería a su proyecto, debe agregar el siguiente código a su archivo `pom.xml`:

```
...
<dependencies>
        ...
    <dependency>
        <groupId>com.github.almasb</groupId>
        <artifactId>fxgl</artifactId>
        <version>0.5.4</version>
    </dependency>
</dependencies>
...
```

[1] `https://github.com/AlmasB/FXGL`

Recuerde mantener separadas las dependencias de su código. Lo que implementó en la tarea 2 corresponde a la lógica del juego y sus modelos, ahora implementará la lógica de interacciones y gráfica, y la interfaz gráfica. Por esta razón, usted no puede tener *imports* de JavaFX ni de FXGL en los paquetes creados para la tarea 2.

Aclaraciones

Usted tiene la opción de hacer uso de *su propio código* implementado en la tarea 2, o de utilizar el código entregado. Se premiará a los estudiantes que utilicen su propio código. En ambos casos pedimos que indiquen en el `readme` qué código base usaron.

Requerimientos adicionales

Además de una implementación basada en las buenas prácticas y patrones de diseño vistos en clases, usted además debe considerar:

- **Cobertura:** Cree los tests unitarios, usando JUnit 4, que sean necesarios para tener al menos un coverage del 90 % de las líneas por paquete. Todos los tests de su proyecto deben estar dentro del paquete `test.java` del proyecto *Maven*. No se contabilizará el coverage del paquete `gui`, lo que significa que no debe testearlo. Probablemente no necesite modificar mucho del código de su tarea 2, pero en caso de implementar los *Bonus*, estos deben ser testeados.
- **Javadoc:** Cada interfaz, clase pública y método público deben estar debidamente documentados siguiendo las convenciones de Javadoc[2]. En particular necesita `@author` y una pequeña descripción para su clase e interfaz, y `@param`, `@return` (si aplica) y una descripción para los métodos. Cláramente la documentación entregada en el código base no cuenta como documentación para usted. Para la parte gráfica, no necesita documentar los métodos, pero sí las clases, entregando una descripción de su propósito.
- **Resumen:** Debe entregar un archivo **pdf** que contenga su nombre, rut, su usuario de Github, un link a su repositorio y un diagrama UML que muestre las clases, interfaces y métodos que usted definió en su proyecto. **No debe incluir los tests, ni el paquete gui** en su diagrama.
- **Git:** Debe hacer uso de git para el versionamiento de su proyecto. Luego, esta historia de versionamiento debe ser subida a Github. Además deberá crear un *tag* "tarea3" en el último commit de su tarea, cómo hacer esto se encuentra en los detalles de entrega.
- **Readme:** En la raíz de su proyecto debe crear un archivo `README.md` con una descripción de su implementación, cómo compilarlo (cómo se usa), dónde se encuentra cada módulo, las *features* que implementó, etc. Más instrucciones en los detalles de entrega.

[2] `http://www.oracle.com/technetwork/articles/java/index-137868.html`

- **Maven:** Su implementación debe estar basada en un proyecto *Maven*, basado en el código de la tarea 2.

Evaluación

- **Código fuente (4.5 puntos):** Se analizará que su código provea la funcionalidad pedida utilizando un buen diseño basado en las buenas prácticas, separando las responsabilidades de su código. Este item se divide en:
  - **Core (1.5 puntos):** corresponde a la implementación de lo pedido anteriormente. Los adicionales al código de su tarea 2, mostrar los elementos en pantalla, las acciones mínimas y las interacciones entre entidades.
  - ***Features* (3.0 puntos):** corresponde a la correcta implementación de 2 *features* mayores y 2 menores, o de 3 *features* mayores.
- **Coverage (0.3 punto):** Sus casos de prueba deben crear diversos escenarios y contar con un coverage de al menos 90 % por paquete. No está de más decir que sus tests deben testear algo (es decir, no ser tests vacíos o sin asserts). Es poco lo que debe cambiar al código de su tarea 2.
- **Javadoc (0.2 puntos):** Cada clase, interfaz y método público debe ser debidamente documentado. Se descontará por cada falta. Solo debe documentar las **clases** en la parte gráfica de su tarea, no necesita documentar los métodos.
- **Resumen y readme (1 punto):** El resumen y readme mencionado en la sección anterior. Se evaluará que haya incluido el diagrama UML de su proyecto. **En caso de no enviarse el resumen con el link a su repositorio, su tarea no será corregida**[3]. En el `README` debe indicar qué *features* decidió implementar y si están completas o incompletas. También debe indicar dónde se encuentran las clases que proveen funcionalidad, controles, colisiones, etc. a su juego.

Entrega

Repositorio: Corresponde al mismo repositorio usado para su tarea 2, **si ya invitó al equipo docente, no debe hacerlo de nuevo**. Las siguientes instrucciones repiten lo pedido en la tarea 2. Su repositorio de Github debe ser **privado**, y debe llamarse **cc3002-breakout**, lo cual significa que si su usuario de Github es *juanpablos*, el link a su repositorio será `https://github.com/juanpablos/cc3002-breakout`. Además, deberán invitar a la cuenta del equipo

[3] porque no tenemos su código.

docente (*CC3002EquipoDocente*) como colaboradores de su repositorio para que podamos acceder a él. Para hacer esto entran a su repositorio, luego a *Settings*, *Collaborators*, ingresan su contraseña y finalmente escriben *CC3002EquipoDocente* en el campo para buscar por usuario, seleccionan *Add collaborator* y nos habrán invitado exitosamente. No se asusten si su invitación no es aceptada, estas serán aceptadas oportunamente cuando su tarea sea revisada.

Entrega por U-Cursos: Usted debe subir a U-Cursos **solamente el resumen en pdf**, con su nombre, rut, usuario de Github, link a su repositorio y diagrama UML. Si su diagrama UML no cabe en el resumen, inclúyalo de todas formas y añada una imagen de él en su repositorio.

Código: El código de su tarea será bajado de Github directamente considerando como entrega el commit que contenga el *tag* "tarea3". Para hacer un *tag* abra su consola y tipee `git tag -a tarea3 -m "Este es el último commit de mi tarea 3 (ponga un comentario distinto a este por favor)"`, el comentario queda a su elección. Esto pondrá el *tag* "tarea3" en el último commit que haya hecho. Para subir su anotación tipee `git push origin tarea3`, o también `git push --tags`.

Readme: Dentro de su repositorio debe incluir un archivo de *markdown* llamado `README.md` que está (estaba) incluido en el código adjunto de la tarea 2. Usted debe dar una descripción de lo que implementó, cómo lo hizo, a grandes rasgos qué patrones de diseño usó, cómo correr su programa, y otras anotaciones que usted encuentre pertinentes. Aquí puede encontrar un *cheatsheet*[4] y un *template*[5]. El template incluye mucha más información que la solicitada, pero así es como debe presentarse un proyecto profesionalmente. Para esta tarea 3 debe además indicar qué *features* decidió implementar, y dónde se encuentran las clases que proveen funcionalidad a su juego.

Plazo: El plazo de entrega es hasta el **22 de diciembre a las 23:59 hrs**. Se bajará su código hasta su *tag* "tarea3" y se verificará que efectivamente la anotación "tarea3" se haya hecho antes de la fecha límite. No se aceptarán peticiones de extensión de plazo, no hay tiempo...

¡Suerte!

[4] `https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet`
[5] `https://gist.github.com/PurpleBooth/109311bb0361f32d87a2`