UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Puesta en marcha de la plataforma de competiciones de programación online Battlecode 2018 - Implementación de los robots.

TRABAJO FIN DE GRADO

GRADO EN INGENIERIA DE SISTEMAS DE TELECOMUNICACIONES

Autor: Cristóbal Barceló Gómez

Director: Dr. Diego Alonso Cáceres

Codirector: Dr. Juan Ángel Pastor Franco

Cartagena, 02 de Abril del 2020

Agradecimientos

Quería agradecer a mis padres por darme la oportunidad de estudiar en la Universidad. Y a los profesores por enseñarme la disciplina que elegí para introducirme en el mundo laboral.

Índice

Ilustraciones

Capítulo 1. Introducción

Los videojuegos aparte de ser una actividad para divertirse también sirven para la educación y para el desarrollo de las competencias psicológicas, así como para la implementación de diversas estrategias del aprendizaje porque fomentan el trabajo en equipo, y desarrollan la capacidad creativa, crítica y comunicativa del individuo. Hay muchos tipos de juegos que se pueden adaptar a las diferentes necesidades del usuario.

Con los años, el tema de los videojuegos va cogiendo más fuerza, sobre todo entre los jóvenes. Con este proyecto se quiere iniciar a los alumnos para que comiencen en el mundo de la programación con la idea de competir contra sus compañeros y divertirse con ellos, y así puedan desarrollar y probar los conocimientos aprendidos en sus estudios en programación.

El juego que se verá en este trabajo es el Battlecode que es un juego de estrategia. Con este juego los alumnos podrán explotar todo su potencial programando la lógica y la estrategia que tendrá cada robot para ganar las batallas a sus compañeros.

1.1 Objetivo del Proyecto.

El objetivo de este proyecto es la puesta en marcha del Battlecode 2018, un juego de estrategia en tiempo real que consiste en enfrentamientos de dos equipos. El combate se hace en el planeta Tierra y si no hay ningún vencedor la batalla final se traslada a Marte. Estos equipos se programan. Un equipo consta de varios robots, factorías o cohetes la utilización de estos depende de la estrategia y la lógica que se quiera utilizar para programar cada equipo.

Para conseguir la puesta en marcha se necesitará utilizar varios programas. Los programas que se van a utilizar son los siguientes:

i. Docker: Es una plataforma para ejecutar contendores. Con la ventaja que aporta de disponer todos los programas necesarios sin la necesidad de tener que instalarlo para el funcionamiento del Battlecode.
ii. Battlecode: Es el juego que se pondrá en marcha para que los alumnos puedan jugar.

iii. Eclipse: Es un entorno de desarrollo integrado (IDE) que se utilizará para crear los equipos mediante código, que luego se introducirá al Battlecode para ver como luchan los equipos entre sí.

En cada capítulo de este proyecto se explicará la instalación y configuración necesaria para que se pueda implementar, crear y visualizar en 3D del combate entre los dos equipos.

1.2 Estructura de la memoria

El desarrollo de este Trabajo fin de grado se divide los siguientes capítulos:

- *Capítulo 2: Docker*. Aquí se expondrá la instalación del Docker y los problemas que pueden suceder mediante la instalación y unos ejemplos de cómo iniciar una aplicación en Docker.
- *Capítulo 3: BattleCode 2018*. Se explicará la puesta en marcha del juego, como añadir un equipo para combatir con otros equipos, y por último se enseñará como visualizar el combate en 3D.
- *Capítulo 4. Implementación de los robots Java*. En este capítulo se hablará de cómo preparar el IDE Eclipse para programar y depurar el código asociado a los equipos.
- *Conclusiones*. En este capítulo se recomienda el juego como una herramienta práctica para comenzare a programar de una forma diferente y divertida, eso sí, recomendado encarecidamente el leer todas y cada una de las instrucciones antes de comenzar a usar Battlecode.
- *Anexo.* En este apartado se explica la instalación del Docker y se muestra el código Java del jugador que se ha creado.

Capítulo 2. Docker

2.1 Qué es Docker.

Docker es un proyecto de código abierto que utiliza contenedores como una nueva forma de empaquetar aplicaciones. Básicamente consiste en la ejecución de sistemas operativos dentro de otros sistemas, a través de lo que se denomina contenedor, el cual es una unidad de software que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y fiable en cualquier sistema operativo en que pueda instalarse Docker.

Una imagen es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, herramientas del sistema, bibliotecas del sistema y configuraciones.

Las imágenes de los contenedores se convierten en contenedores en tiempo de ejecución, es decir, cuando una imagen se transforma en un contendor cuando se ejecuta. También si un contenedor ha sufrido alguna modificación se podrá convertir en una imagen para su reutilización.

La diferencia que hay entre los contenedores y las máquinas virtuales es que los contenedores virtualizan el sistema operativo en lugar del hardware. Además, los contenedores son más portátiles y eficientes, es decir, una máquina virtual necesita virtualizar un sistema operativo para hacer funciona la aplicación que se desear ejecutar, pero en cambio un contenedor no. El contenedor coge las funciones más básicas del sistema operativo del ordenador donde se ejecutará la aplicación. De esta manera la aplicación ya tiene todo lo necesario dentro del contenedor y así no se tiene que instalar otro sistema operativo.

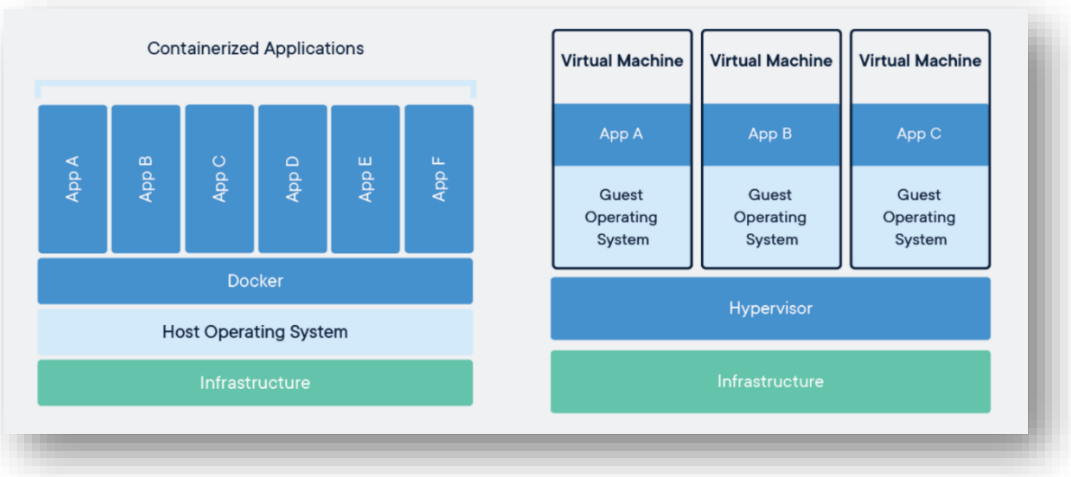

Figura 1: Contenedores y Máquinas Virtuales

La tecnología de Docker es única porque se centra en los requisitos de los desarrolladores para separar las dependencias de las aplicaciones de la infraestructura.

Muchas empresas están optando por encapsular sus aplicaciones o microservicios para que se ejecuten sobre una máquina virtual, para después subirlo a la nube. Estás empresas están utilizando los contenedores para mover aplicaciones a la nube porque una vez que una aplicación se encapsula en un contenedor, es más fácil comenzar a separarla en varios microservicios.

La utilización de Docker permite de una manera sencilla la creación, virtualización y el traslado de contenedores de un sistema a otro, lo cual está generando un gran avance tecnológico en la utilización de la nube.

Estos microservicios se pueden estandarizar en contenedores y proporciona un despliegue más rápido a la hora de implementar en la nube y evitar conflictos entre paquetes. Además, como los contenedores son sencillos de desplegar, pueden ser usado por usuarios sin grandes conocimientos de informática. La facilidad que proporciona Docker para despliegue de contendores de forma rápida y eficiente le convierte en una buena herramienta para utilizarla en la nube.

2.2 Ejecución de contenedores

A continuación, se verán unas series de comandos que indican cómo se ejecuta un contendor de Docker. No se podrán interactuar con todos los contenedores, solo con aquellos que estén preparados para ello. Se utilizarán para el ejemplo, los contenedores Ubuntu y Hello-World.

Para ver las imágenes que están disponibles para descargar se irá al repositorio del Docker (https://hub.docker.com/search?q=&type=image).Para ejecutar Ubuntu se seguirá estos pasos:

Primero se tendrá que escribir este comando para descargar la imagen de Ubuntu: “Docker pull Ubuntu”, como se observa en la Figura 2.

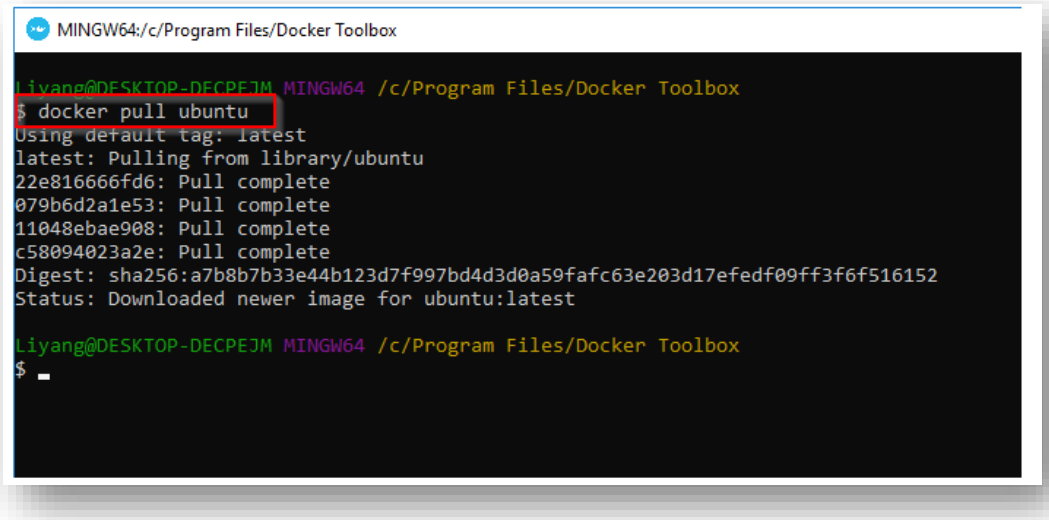

Figura 2: Descarga Ubuntu

Para poder empezar a ejecutar esta imagen de Ubuntu e interactuar con el contenedor que se crea a partir de esta imagen se escribirá el siguiente comando:

“`docker run -it ubuntu`”.

Con este comando se creará un contenedor de Ubuntu y se nos habilitará la consola de Ubuntu para poder escribir cualquier comando de Linux. En este ejemplo se han introducido los comandos “ls” y “ls -l” que muestran todos archivos y carpetas del directorio del contenedor como se puede visualizar en la Figura 3.

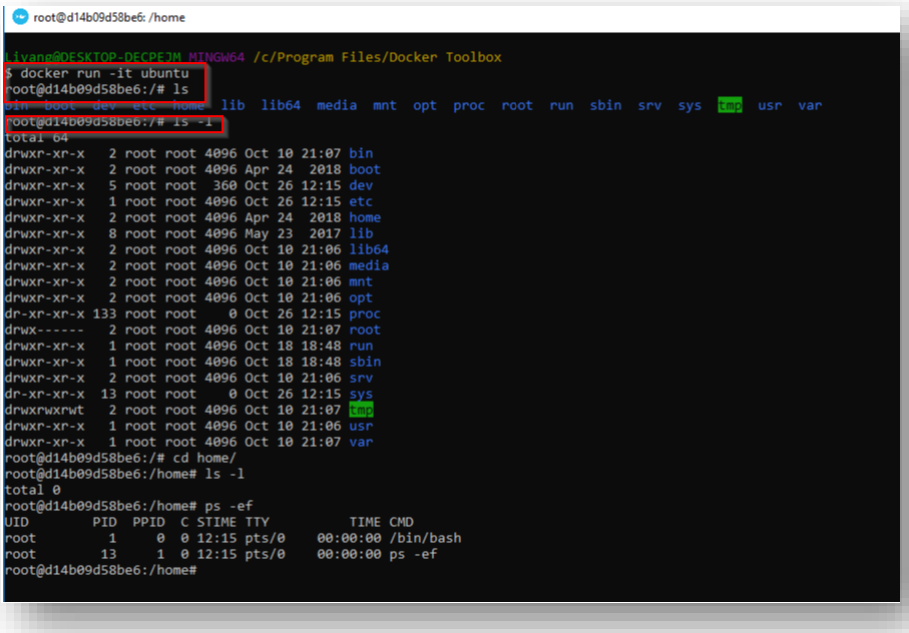

Figura 3: Ejecutar Ubuntu

En el ejemplo anterior se ha visto el comando para ejecutar Ubuntu y poder interactuar con él, pero no con todos los contenedores se podrá interactuar. En la figura 4 se muestra un ejemplo de cómo ejecutar un contenedor sin interactuar con él. Como se ve en la Figura 4 se ha introducido el comando:

"`docker run hello-world`"

y el resultado son mensajes de información sobre Docker.

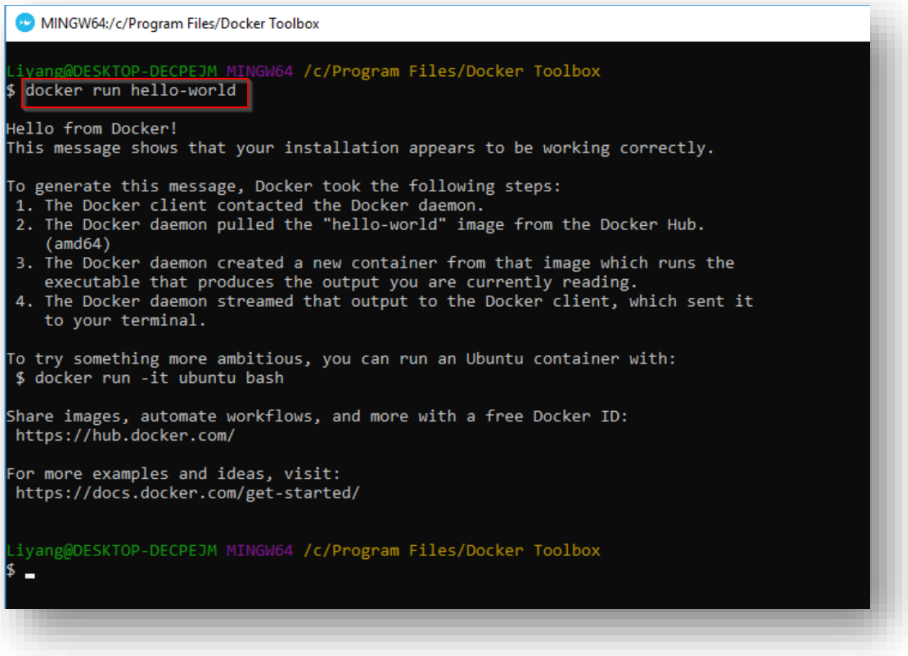

Figura 4: Ejecutar Hello-world

Para ver las imagenes que se tienen en nuestro sistema se utilizará el comando "docker images" (Figura 5).

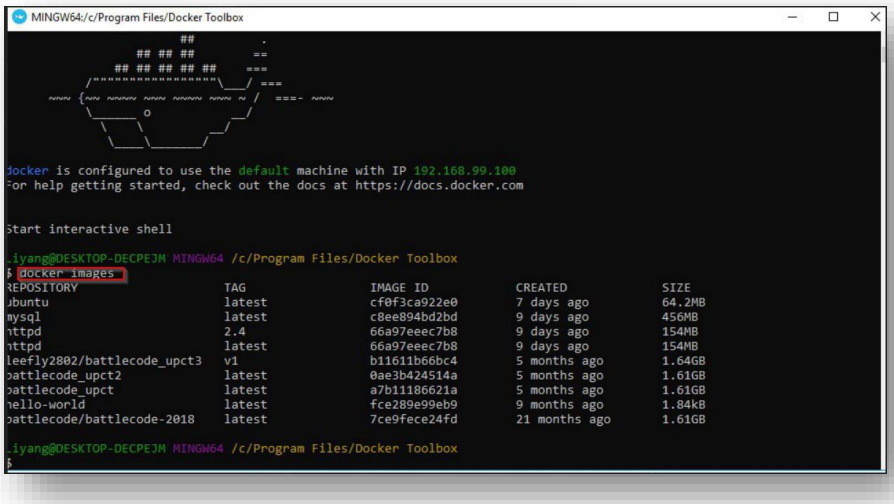

Figura 5: Imágenes Docker

Para ver los contenedores se utilizará el comando:

“docker ps -a”

Y para visualizar solo los contenedores que hay activos se escribirá el comando:

”docker ps”.

Como se muestra la Figura 6

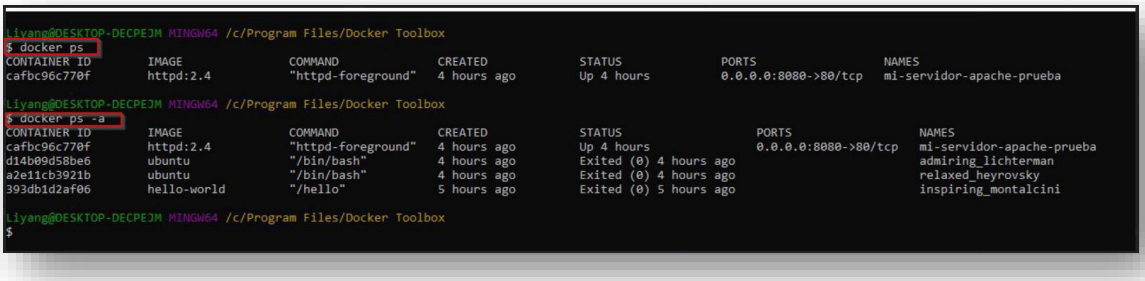

Figura 6:Contenedores Docker

Capítulo 3. Battlecode 2018

3.1 Que es Battlecode

Battlecode 2018 es un juego de estrategia en tiempo real que consiste en el enfrentamiento de dos equipos. El enfrentamiento se lleva a cabo en la Tierra y si no se decide el vencedor, mientras se está destruyendo el planeta, la batalla final será en Marte, que es el planeta elegido para trasladarse la humanidad por el deterioro del planeta Tierra.

La creación de estos equipos se hace mediante la programación, en Python, Java o C. Cada equipo consta de un ejército, y según la estrategia que tenga el jugador puede combinar una serie de robots, factorías (Factories) y cohetes (Rockets) como le convenga para sus planes de combate.

Los robots pueden ser:

- Workers: la funcionalidad de este robot es cosechar los minerales (Karbonite), crear estructuras y repararlas.
- Knights: Son robots para luchar cuerpo a cuerpo y se crean en las Factories.
- Rangers: Son unidades que atacan a distancia y se crean en las Factories.
- Mages: Atacan a distancia y son expertos en causar daño a grandes áreas y se crean en las Factories.
- Healers: Son robots de apoyo que curan a sus aliados para que puedan seguir luchando.

Las estructuras son:

- Rockets: Es la única unidad que sirve para viajar a otros planetas, es el vehículo de transporte.
- Factories: Las factorías se utilizan para crear robot que luchan contra los enemigos.

La versión de Battlecode que se va a utilizar es la de 2018. Los datos de los robots y las estructuras junto a las especificaciones del juego se pueden ver en el siguiente enlace (https://s3.amazonaws.com/battlecode-2018/specs/battlecode-specs-2018.html).

3.2 Puesta en marcha

Para comenzar con la puesta en marcha, se deberá tener Docker instalado y funcionando correctamente, si no el contenedor del Battlecode no funcionará.

1. Descarga.

Se descargará Battlecode2018 de este enlace (https://github.com/battlecode/bc18-scaffold) seleccionando las opciones señaladas en rojo de la Figura 7. En el enlace anterior también se explica cómo se instala Battlecode para los diferentes sistemas operativos.

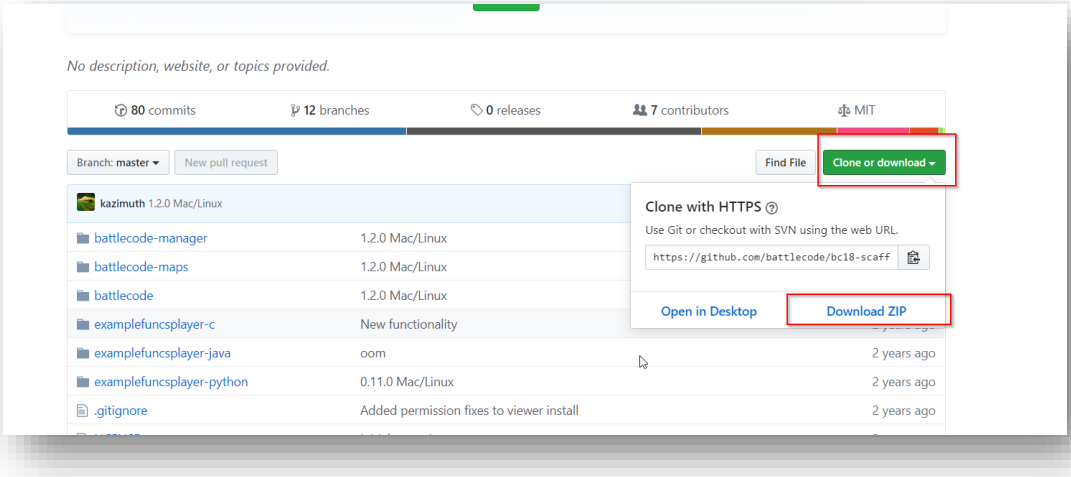

Figura 7: Descarga Battlecode 2018

Una vez descargado, se descomprimirá y se verá los archivos que muestra la Figura 8.

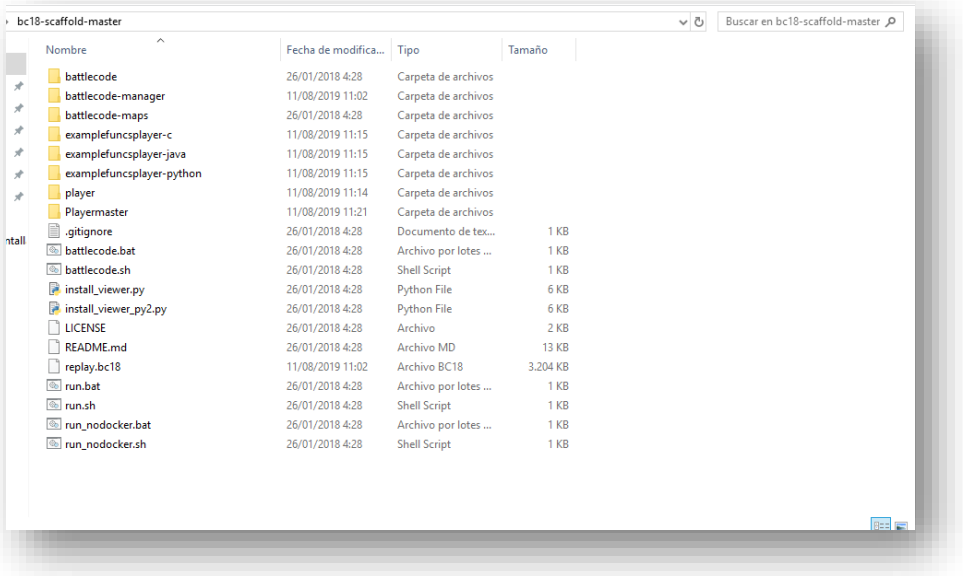

Figura 8: Archivos de Battlecode 2018

2. Ejecutar Battlecode

Para ejecutar Battlecode en Docker primero se copiará la ruta de la carpeta donde se haya guardado y después se harán las siguientes sustituciones de caracteres.

Se sustituirá los caracteres “\” y ”:” por este “/”, y al principio de la ruta también se pondrá el carácter “/”.

Por ejemplo, si tiene esta ruta:

`C:\Users \Desktop\bc18-scaffold-master`,

con la sustitución de los caracteres se quedaría así:

`/C/Users/Desktop/bc18-scaffold-master`.

Cuando se haya sustituido los caracteres, el comando completo para poner en la consola Docker seria:

```
cd C:\Users\Legion\Desktop\bc18-scaffold-master
```

Este comando nos llevará a la carpeta donde se tendrá descomprimido el Battlecode (Figura 9).

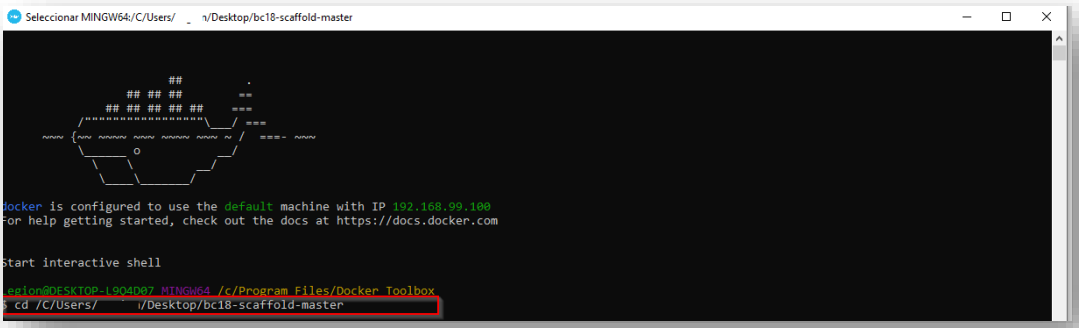

Figura 9: Ejecutar Docker (1)

Cuando se esté dentro de la carpeta, se meterá el siguiente comando: “bash run.sh”. Este comando ejecutará Battlecode.

Saldrá una opción que dirá si se quiere continuar y se escribirá la letra “Y”. La primera vez que se ejecute el Battlecode se descargan muchos archivos: se dejará que descargue todo. Cuando la descargar haya finalizado saldrá una dirección web como se observa en la Figura 10.

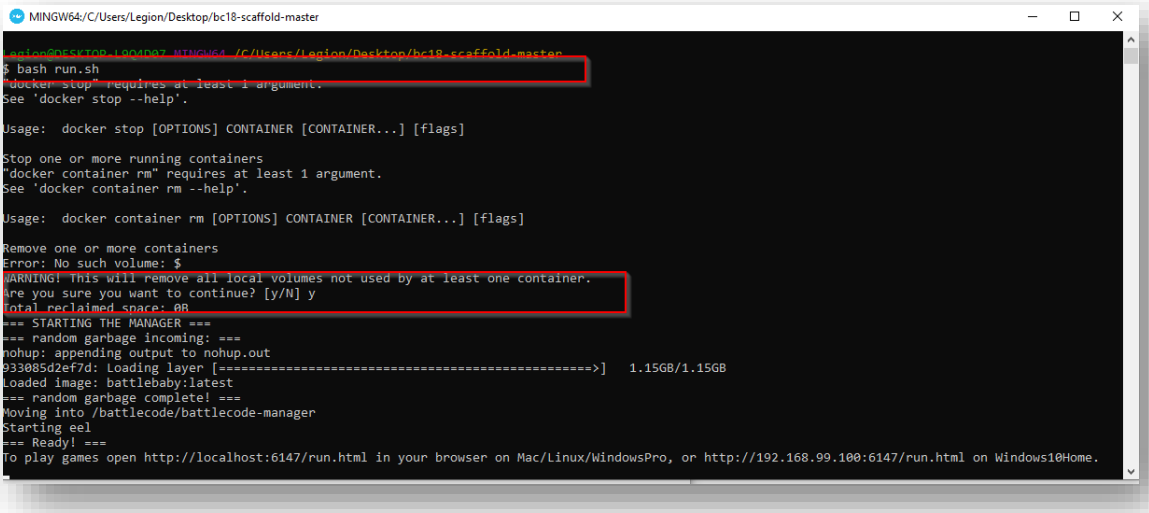

Figura 10: Ejecutar Battlecode 2018

Se copiará la dirección web http://192.168.99.100:6147/run.html. Esa dirección se pegará en el navegador y saldrá la ventana que se muestra en la Figura 11.

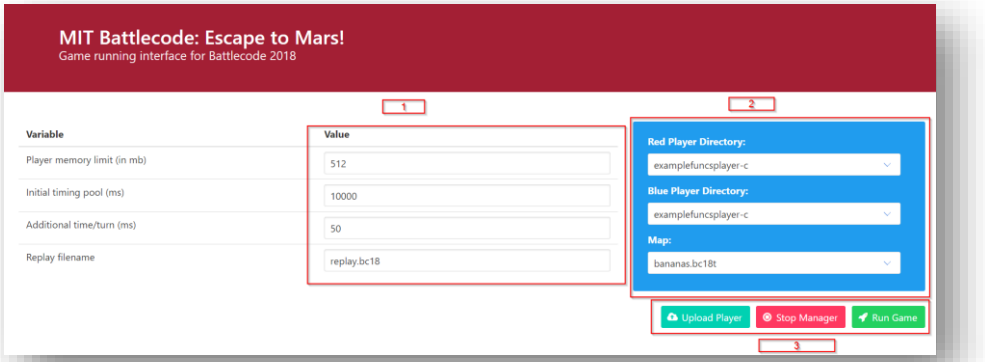

Figura 11:Navegador

El recuadro 1 de la Figura 11 muestra los parámetros con los que se va a ejecutar Battlecode. En el recuadro 2 es donde se seleccionan los jugadores y el mapa. Y en el recuadro 3 se tendrán los botones para jugar, parar el Battlecode y subir jugador.

3.3 Añadir Jugadores

En este apartado se mostrará cómo se pueden introducir jugadores en el Battlecode.

Como los equipos se pueden programar en tres lenguajes que son Python, C y Java, aquí se explicará cómo introducir los jugadores a Battlecode en Java. En los otros lenguajes de programación, se procederá de igual manera, pero con los archivos correspondientes a cada lenguaje.

Para introducir un nuevo jugador en Battlecode, primero se irá a la carpeta del juego y se buscará la carpeta con el nombre examplefuncsplayer-java (Figura 12).

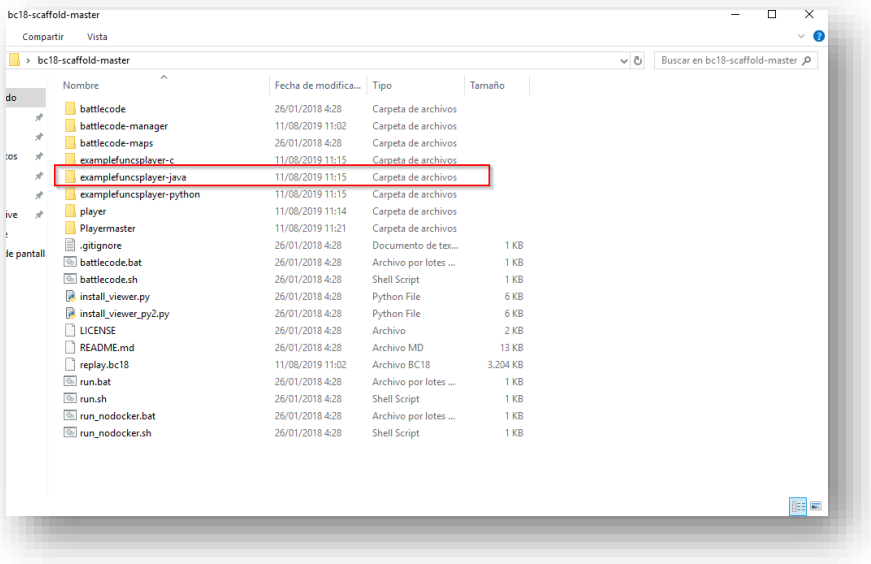

Figura 12: Añadir Jugador (1): Búsqueda de la carpeta.

Dentro de la carpeta examplefuncsplayer-java, se copiarán los dos archivos seleccionados como muestra la Figura 13.

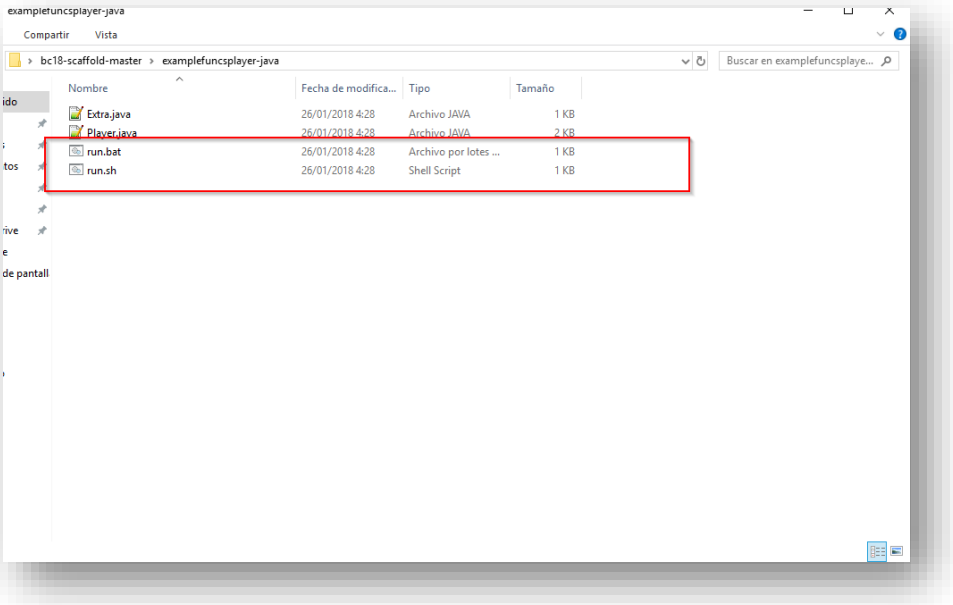

Figura 13: Añadir Jugador (2): copia de archivos seleccionados.

Una vez copiados esos dos archivos, se volverá a la carpeta raíz del Battlecode y se creará una carpeta que en este ejemplo se llamará Jugador_Prueba (Figura 14).

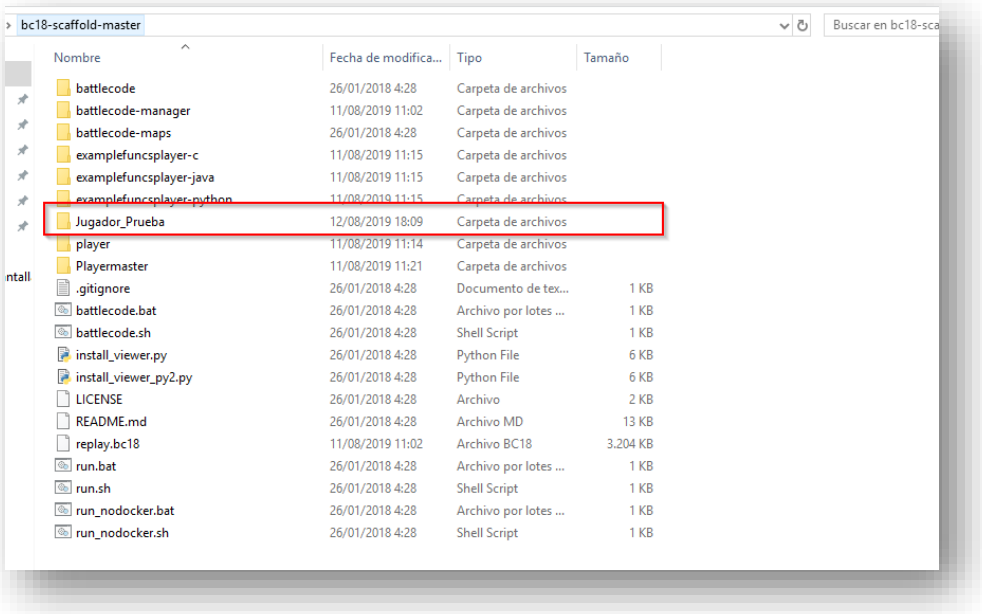

Figura 14: Añadir Jugador (3): crear carpeta

Dentro de la carpeta Jugador_Prueba se pegarán los dos archivos seleccionados (Figura 15) y todos los archivos ".java" del jugador que se vaya a crear. Debe tener la misma estructura que en la carpeta examplefuncsplayer-java.

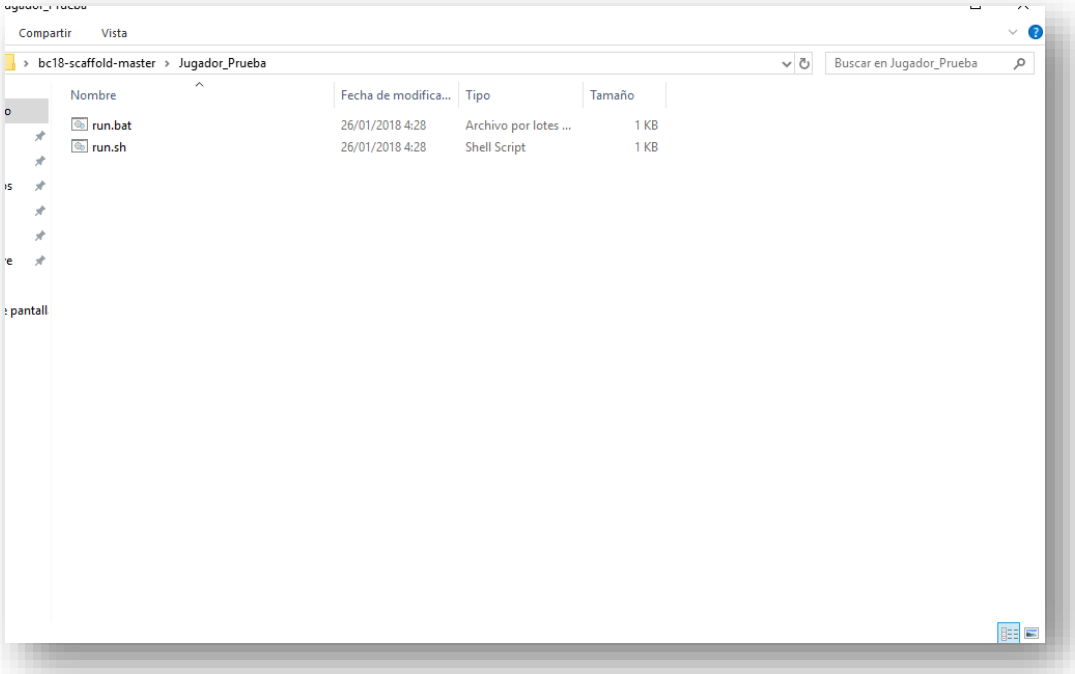

Figura 15: Añadir Jugador (4): copiar archivos

Una vez pegados esos dos archivos, se ejecutará Battlecode y en el desplegable donde se seleccionará los jugadores saldrá el nuevo jugador creado (Figura 16).

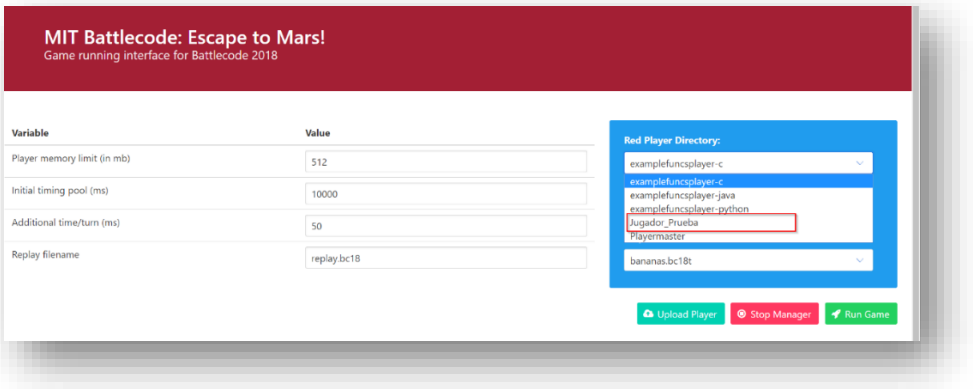

Figura 16: Visualizar carpeta en el navegador

3.4 Visualización 3D

Para poder verlo en 3D se tendrá que haber instalado Python, como ya se comentó en el primer paso de la instalación del Docker. La primera visualización del Battlecode es en 2D, para poder visualizarlo en 3D se seguirán los siguientes pasos.

En la carpeta del Battlecode, se ejecutará el archivo install_viewer.py (Figura 17).

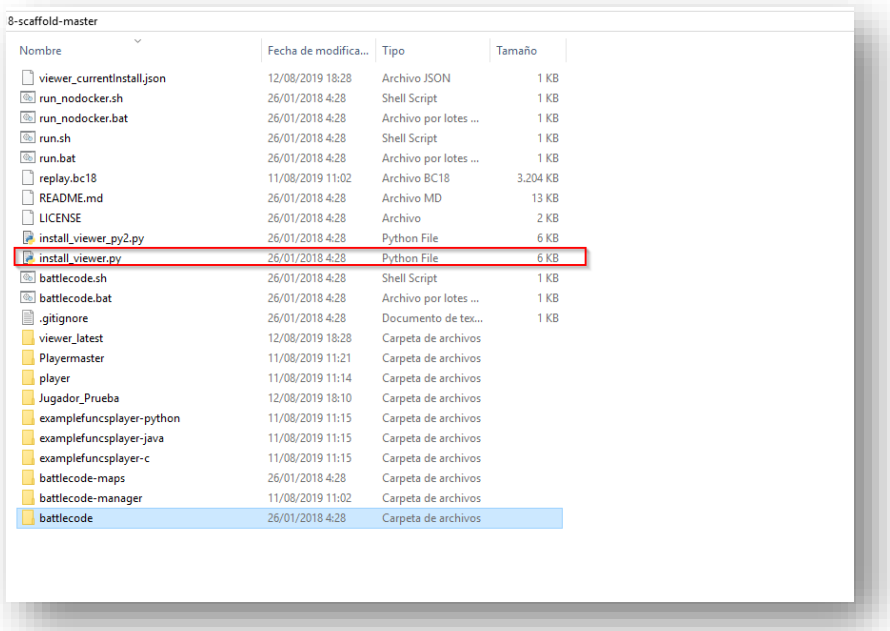

Figura 17: Battlecode visualización 3D (1)

Una vez ejecutado saldrán diversas opciones. En nuestro caso como se tiene instalada la versión Windows 64 bits se ha marcado la opción 1 y una vez marcada esa opción (se ha escrito la letra “Y”), empezará con la instalación (Figura 18).

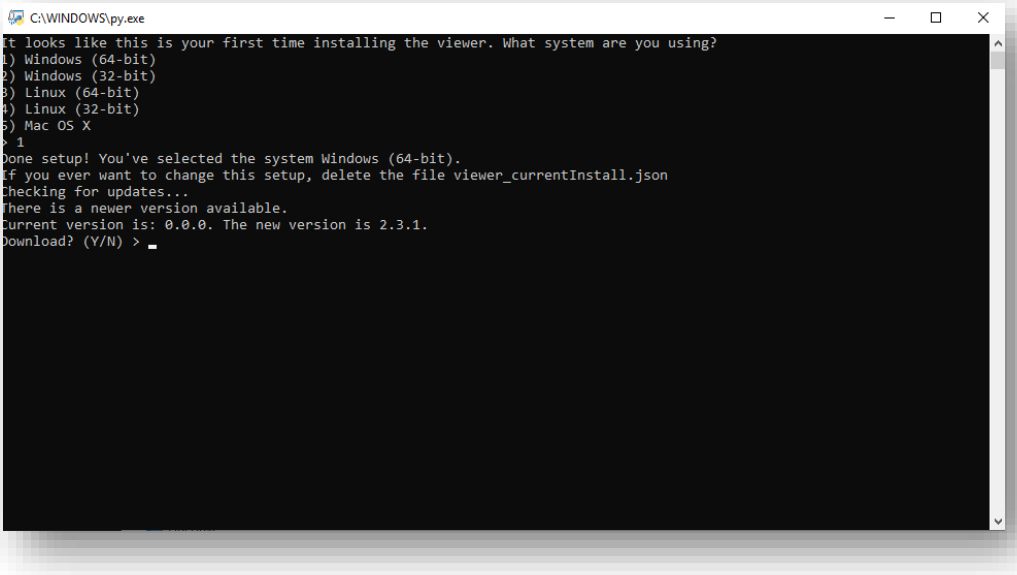

Figura 18: Ejecutar archivos viewer.py

Cuando termine la instalación saldrá en la carpeta del Battlecode una carpeta nueva llamada viewer_latest (Figura 19).

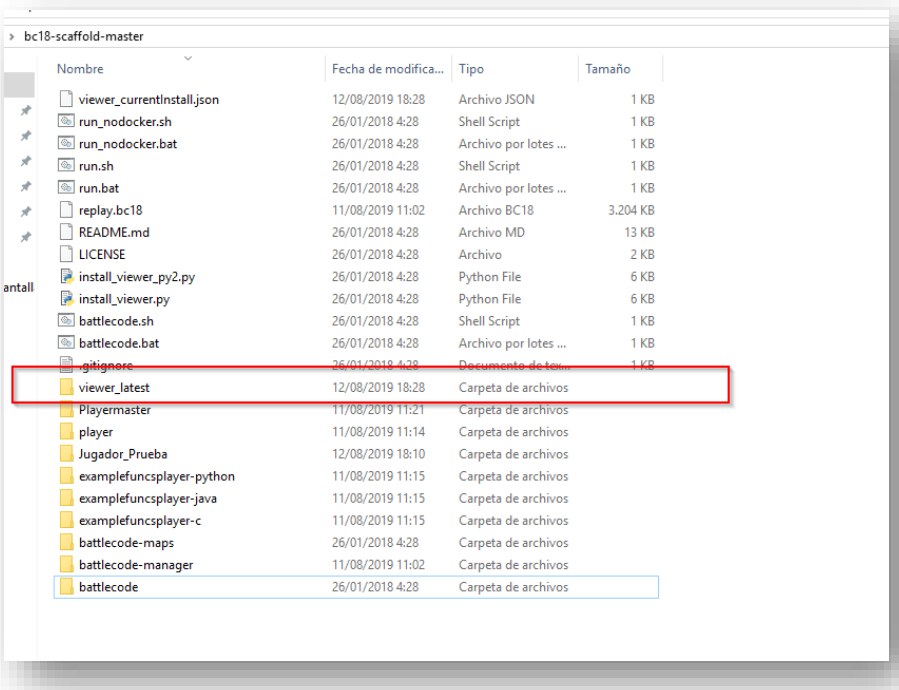

Figura 19: Carpeta creada viewer_lasted

Antes de meterse en esta carpeta, se tendrá que generar el archivo que después se necesitará para poder visualizarlo en 3D.

Para generar este archivo se arrancará Docker, se ejecutará Battlecode y se irá al navegador web. Se pondrá un nombre al fichero, este fichero es el que se va a visualizar en 3D, por ejemplo, en este caso se llamará replay.bc18. No debemos olvidar, siempre se debe de poner la extensión .bc18. Una vez puesto el nombre del fichero, se seleccionarán los jugadores y se comenzará a jugar (Figura 20).

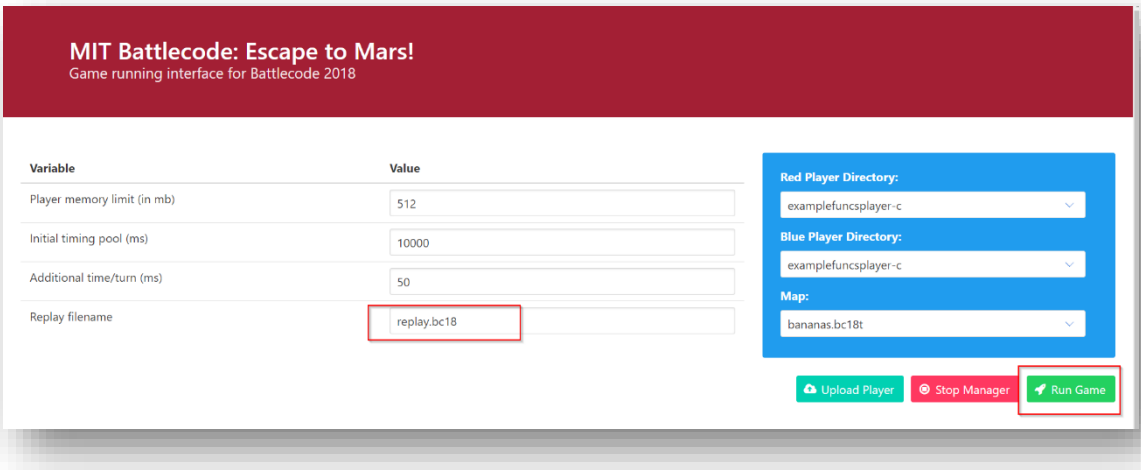

Figura 20: Battlecode visualización 3D (4): Guardar partida

Aparecerá la siguiente pantalla que se corresponde con la visualización del juego en 2D (Figura 21).

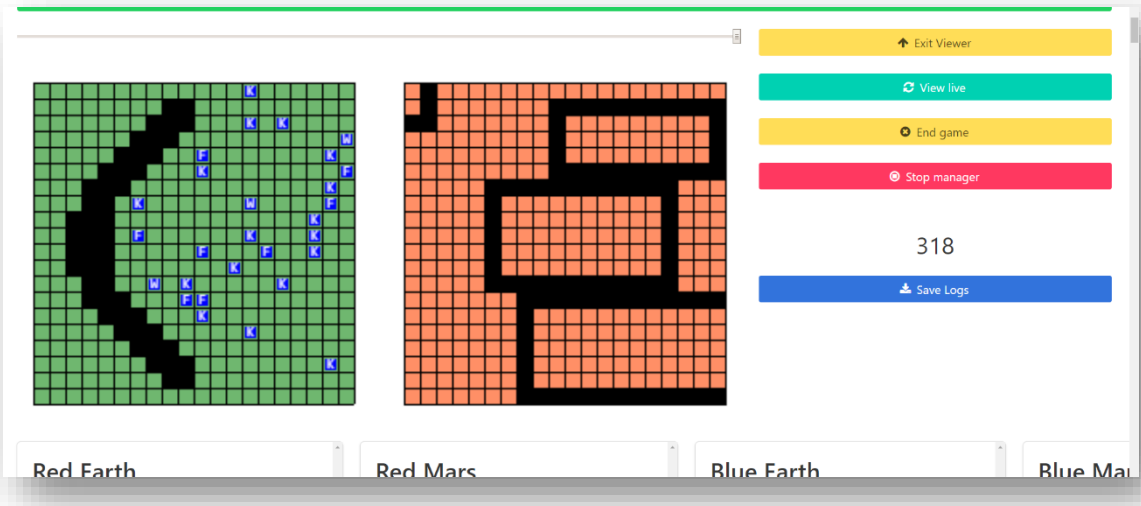

Figura 21: Battlecode visualización 3D (5)

Cuando haya terminado la partida. Se volverá a ir a la carpeta del Battlecode y se habrá creado un fichero con el nombre que se haya puesto (Figura 22).

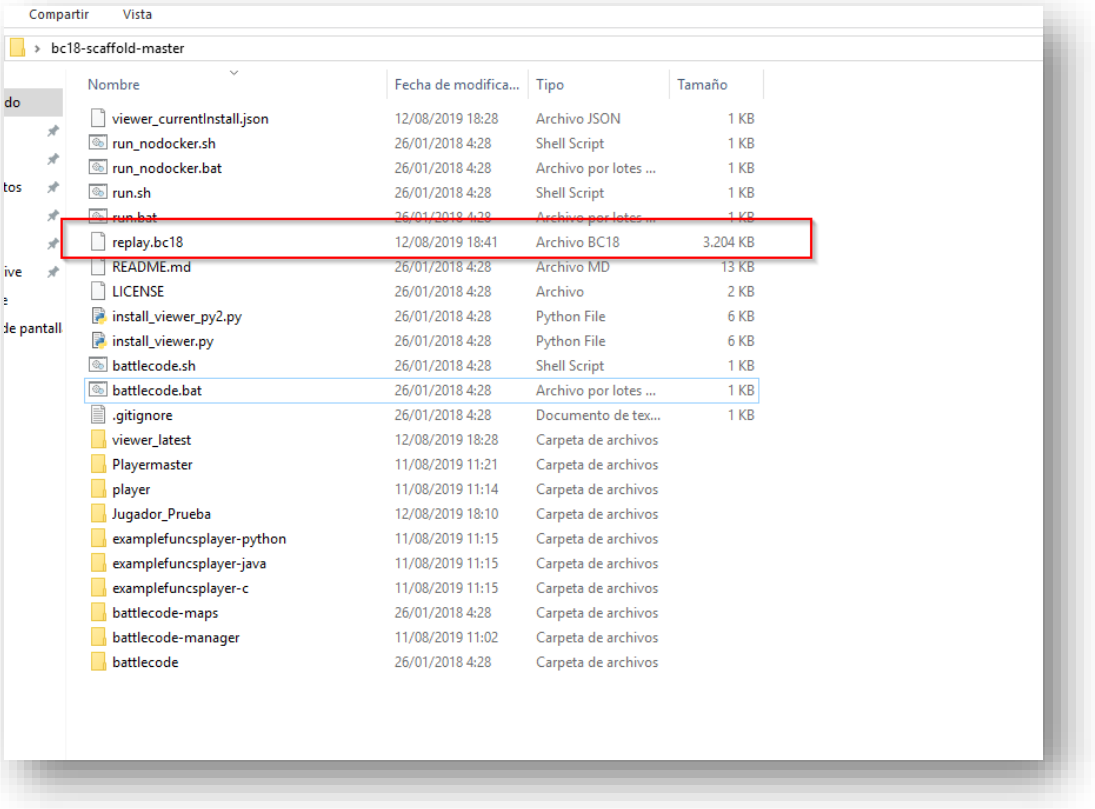

Figura 22: Battlecode visualización 3D (6): Fichero creado de la partida

Ahora sí, en la carpeta viewer_latest, se arrancará el Battleclient. Saldrá una configuración inicial, se seleccionará los parámetros adecuados para nuestro ordenador.

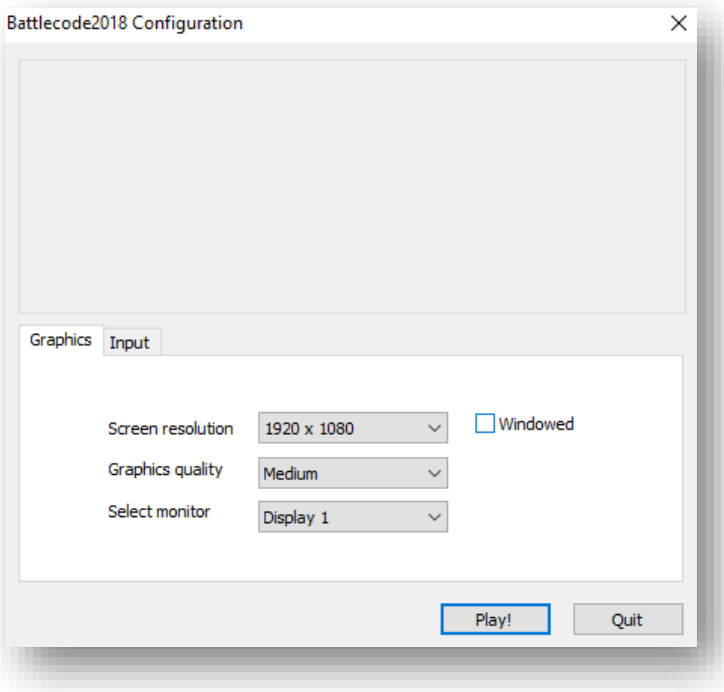

Figura 23: Battlecode visualización 3D (7): Battleclient

Una vez seleccionados los parámetros, se nos abrirá una ventana como en la Figura 24. Se pinchará en el icono de arriba, el de las tres rayas, y saldrá una ventana donde se buscará nuestro archivo que se ha generado. Una vez seleccionado el archivo hay que darle al botón Select.

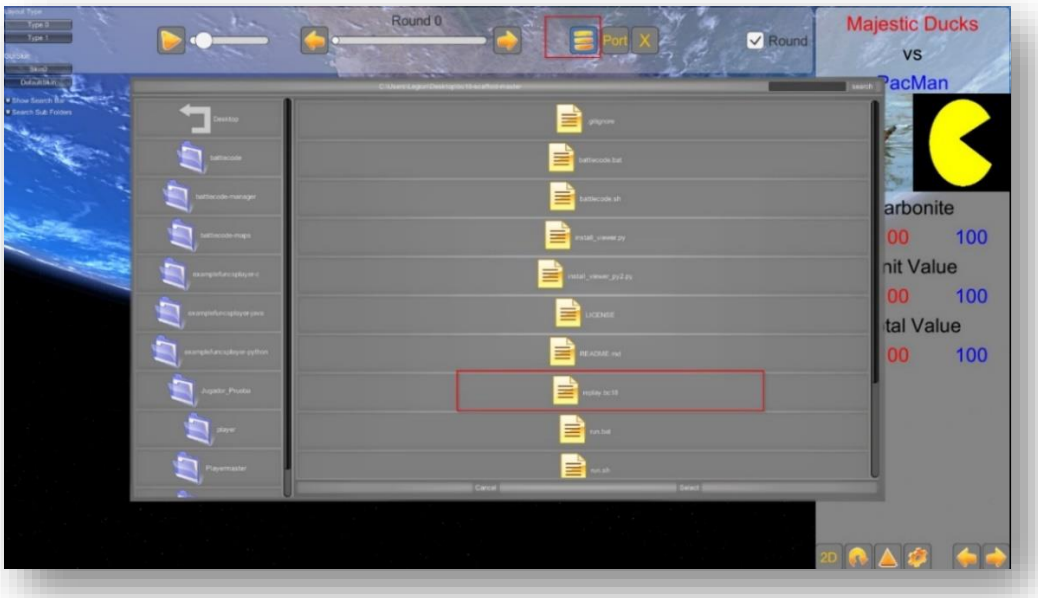

Figura 24: Battlecode visualización 3D (8): Buscar fichero de la partida

Aquí se podrá visualizar la partida en 3D, para empezar a visualizar se le dará al play que está arriba a la izquierda (Figura 25).

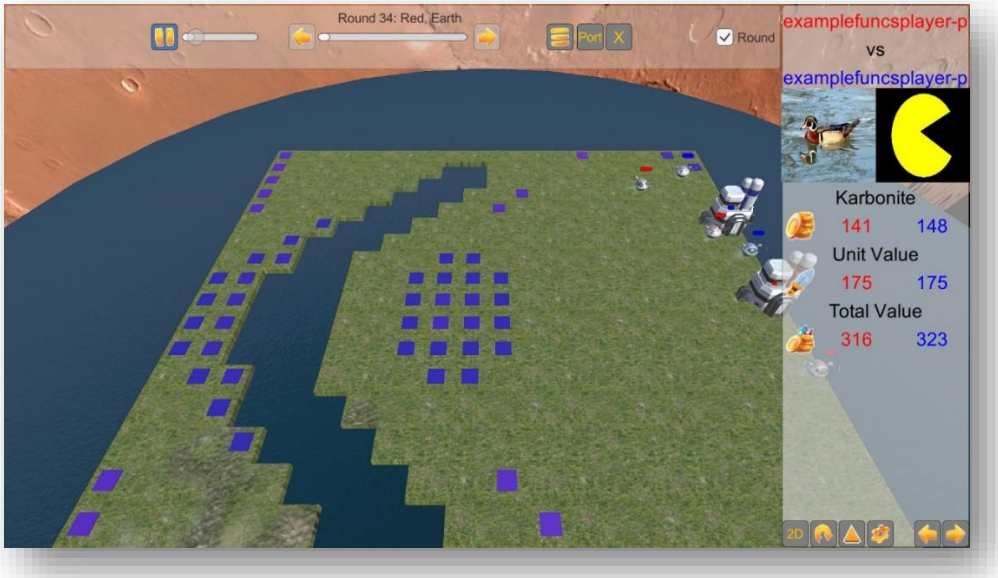

Figura 25: Battlecode visualización 3D (9): Visualización de la partida creada

Para cerrar el programa hay que darle a la tecla ESC.

Capítulo 4. Implementación de los robots Java

4.1 Introducción

El Battlecode 2018 se puede programar en 3 lenguajes, C, Python o Java. En este apartado se explicará cómo crear un equipo con el lenguaje de programación Java.

Cada jugador lleva un equipo que puede constar de robots, cohete o factorías y con el uso de estos recursos formará sus estrategias.

Para programar se puede utilizar un editor de texto o un entorno de desarrollo (IDE), aquí se explicará como programarlo con un IDE, en este caso será Eclipse. Se hará así porque el propio IDE ayuda a la hora de programar.

4.2 Eclipse

Para descargar Eclipse se hará desde su página oficial (https://www.eclipse.org/ide/).

En este apartado se mostrará cómo preparar Eclipse para poder programar un jugador en Battlecode 2018. Para comenzar, se empezará creando una carpeta en nuestra carpeta del Battlecode, por ejemplo, se llamará Jugador_Eclipse (Figura 26).

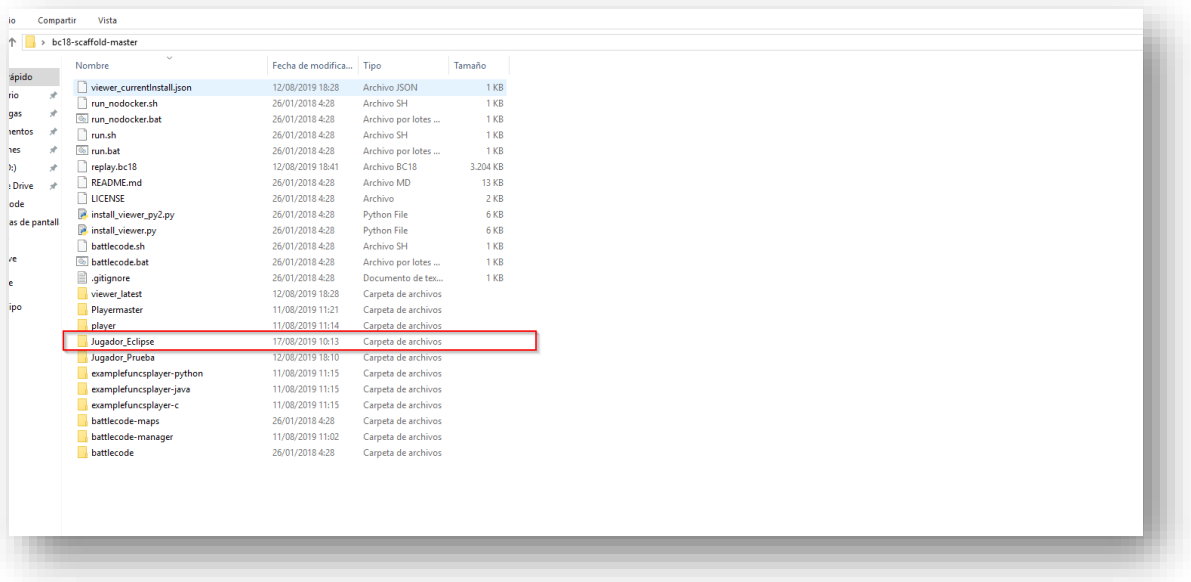

Figura 26: Creación de carpeta eclipse

Una vez descargado e instalado Eclipse, se ejecutará. Una vez abierto como muestra la Figura 27 se creará un proyecto nuevo.

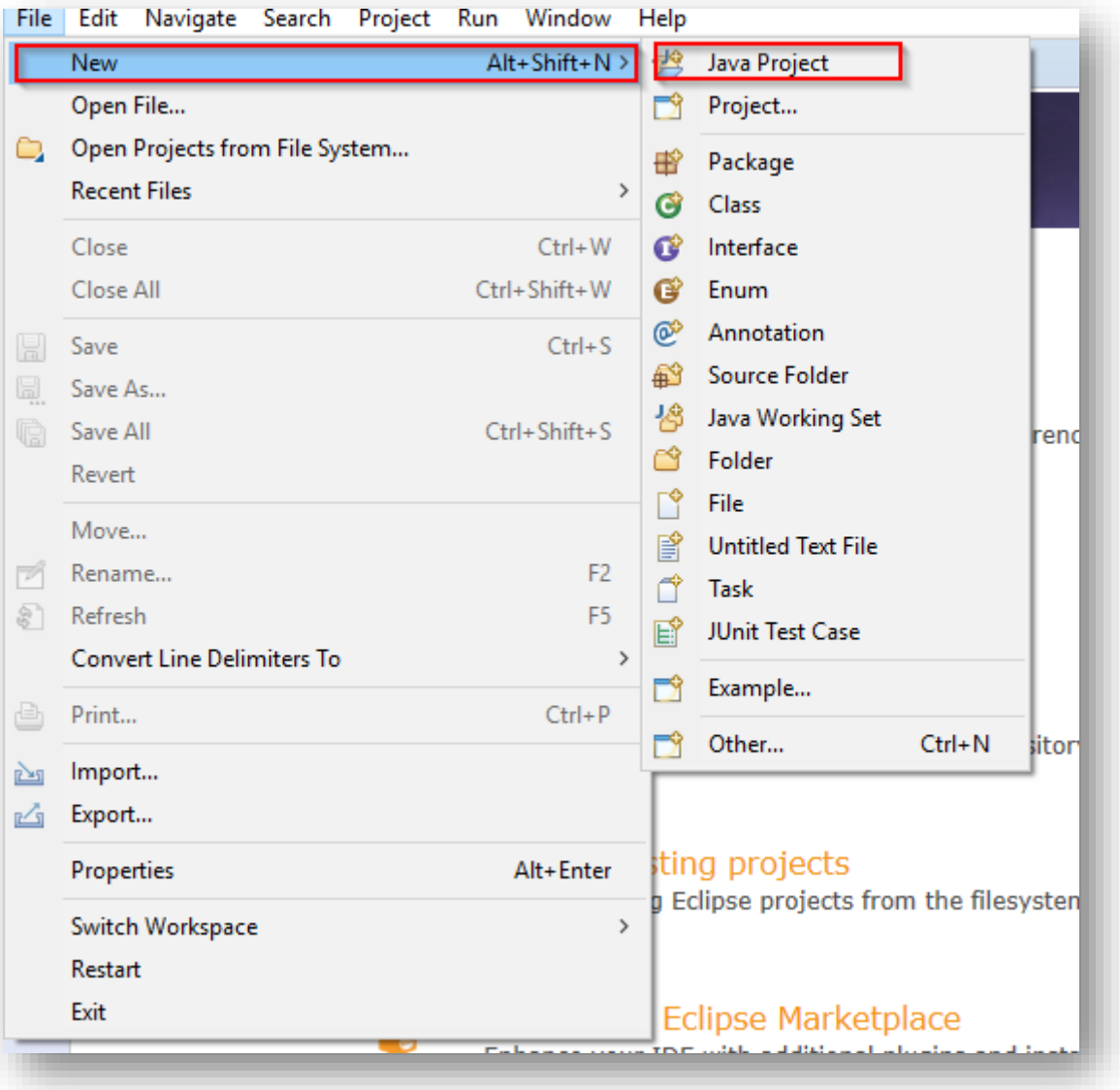

Figura 27: Eclipse (2): Crear un proyecto

En la Figura 28 se podrá ver que se pone el directorio donde se ha creado nuestra carpeta de Jugador_Eclipse y una vez configurado el proyecto se dará a Finish, concluyendo con la creación del proyecto.

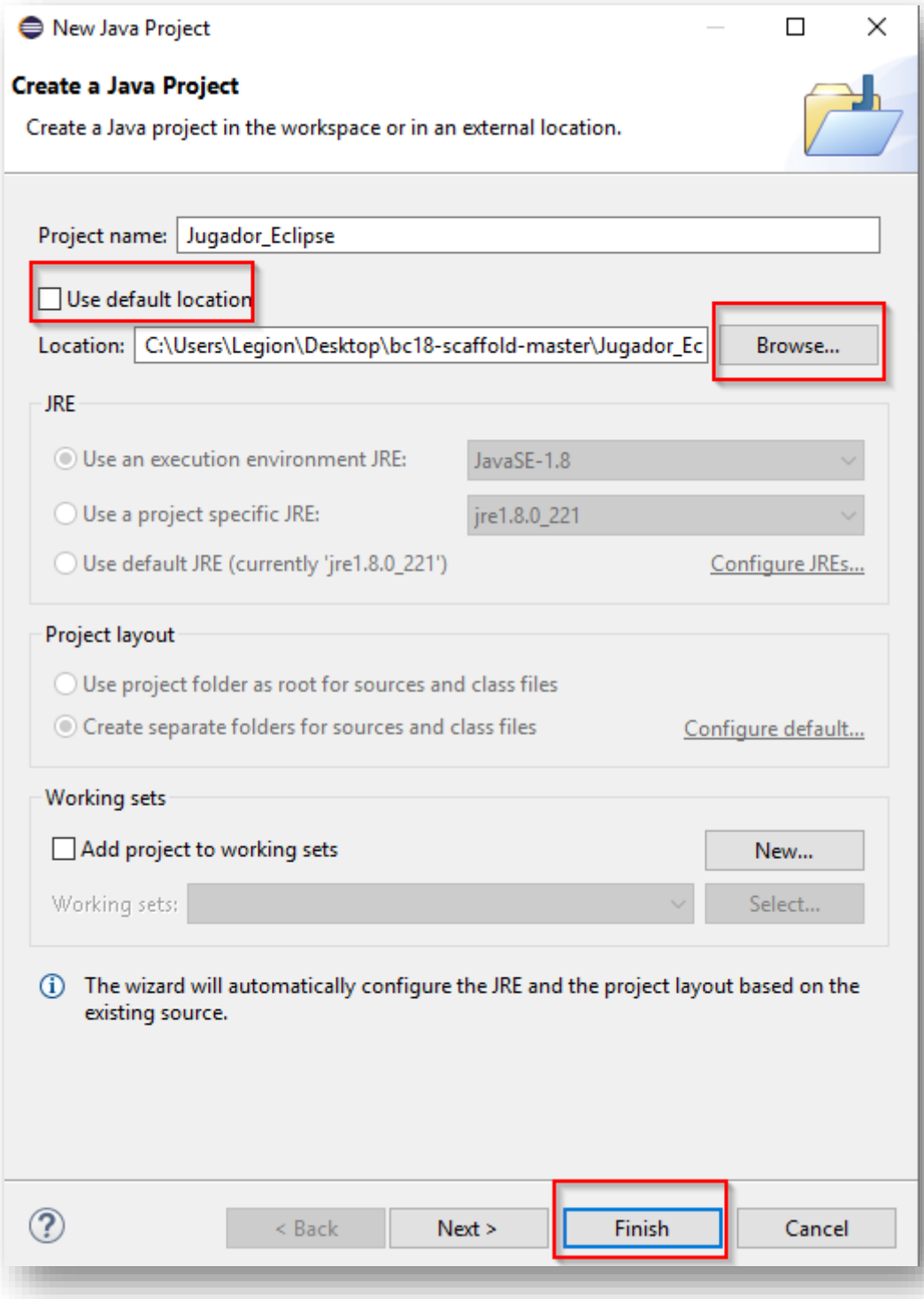

Figura 28:Eclipse (3): Parámetros de creación de un proyecto

A continuación, se creará una clase, se tendrá que crear como package default como se muestra en las siguientes dos Figuras.

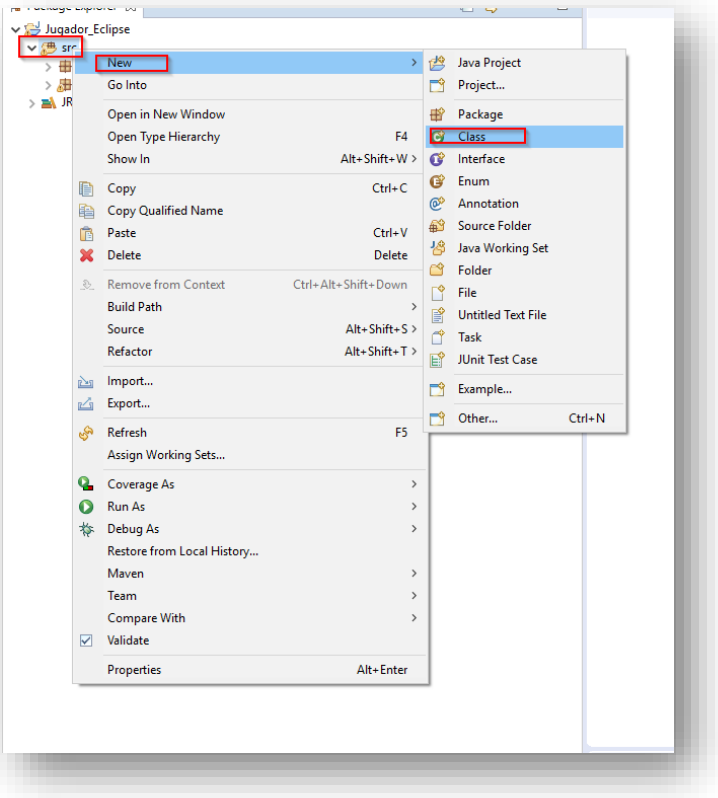

Figura 29:Eclipse (4): Crear clase

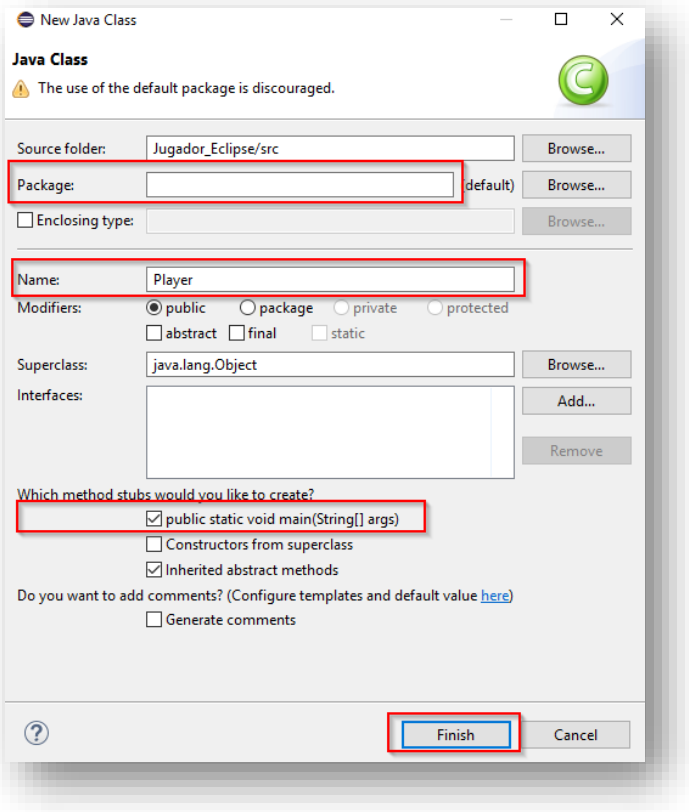

Figura 30: Eclipse (5): Crear clase

Se tendrán que transportar las librerías. Siguiendo las Figuras se llegará a la carpeta bc. Esa carpeta se copiará y se pegará en nuestro proyecto de eclipse.

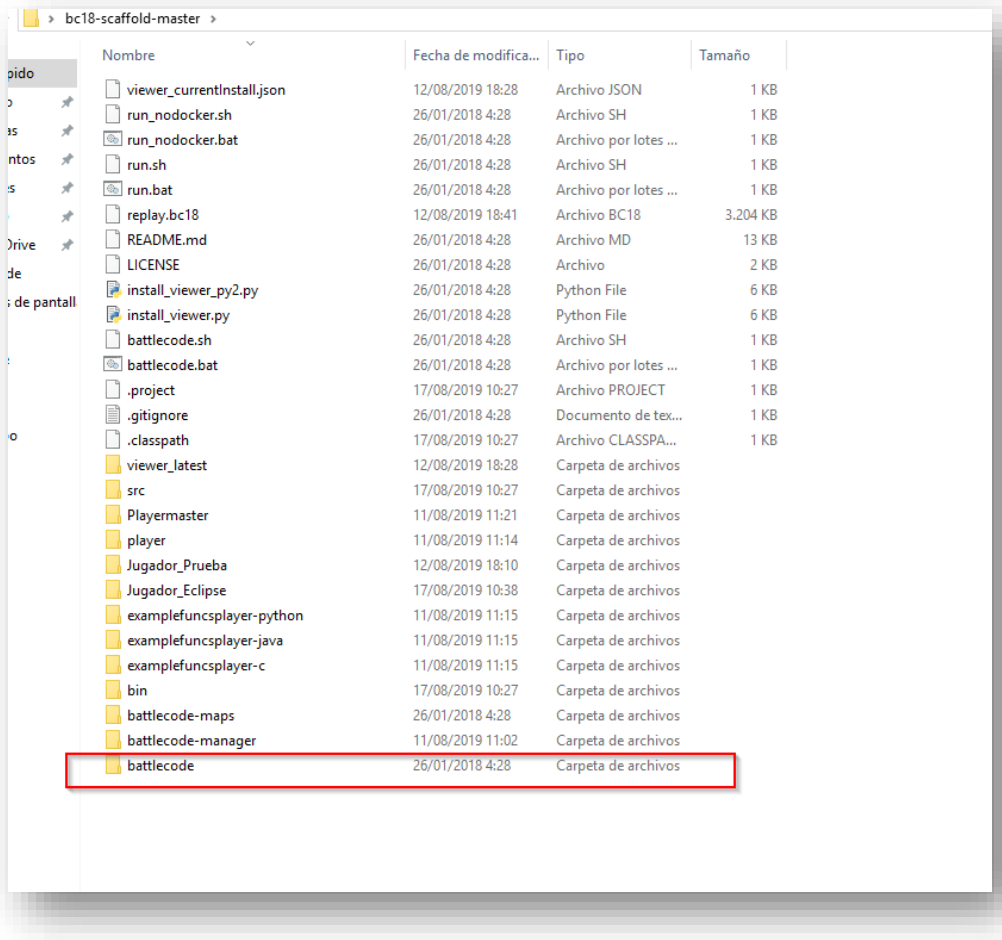

Figura 31: Eclipse (6): Carpeta Battlecode

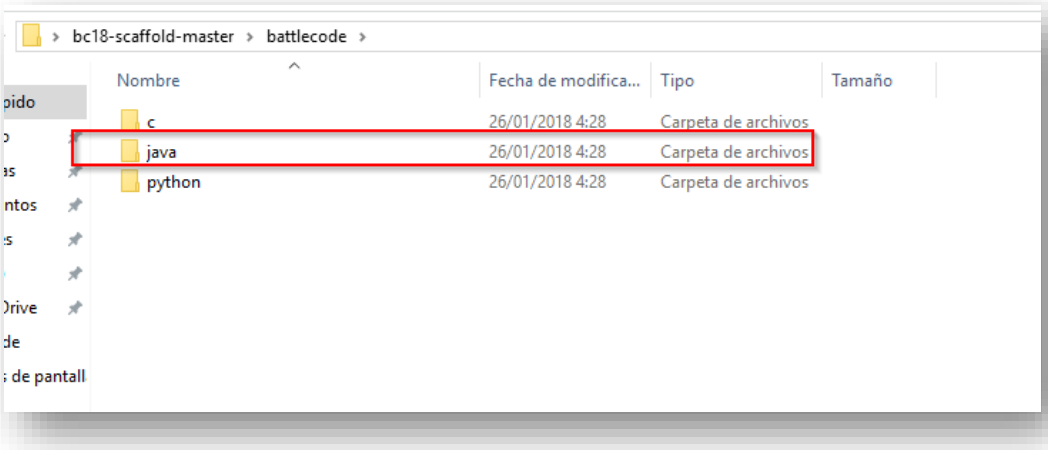

Figura 32: Eclipse (7): Carpeta java

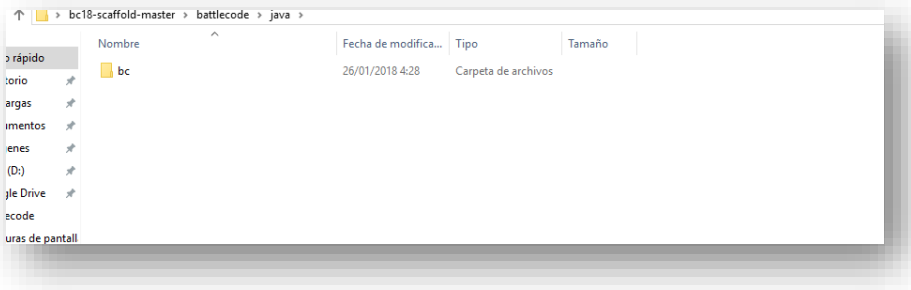

Figura 33: Eclipse (8): Carpeta BC

Como se puede observar en la Figura 34 la estructura del proyecto debe de quedarse así.

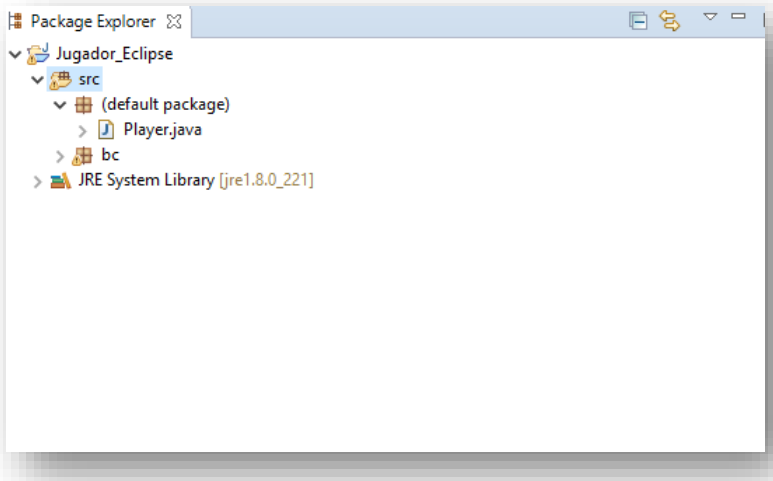

Figura 34: Eclipse (9): Estructura del proyecto

Hay que tener la opción Build Automatically activada como se muestra en la Figura 35.

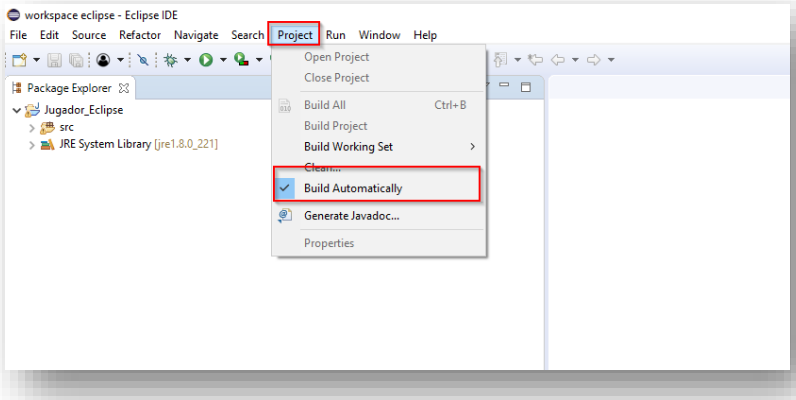

Figura 35: Eclipse (10): Build Automatically

Para finalizar se irá a la carpeta de examplefuncsplayer-java y se copiará los dos archivos run.sh y run.bat.

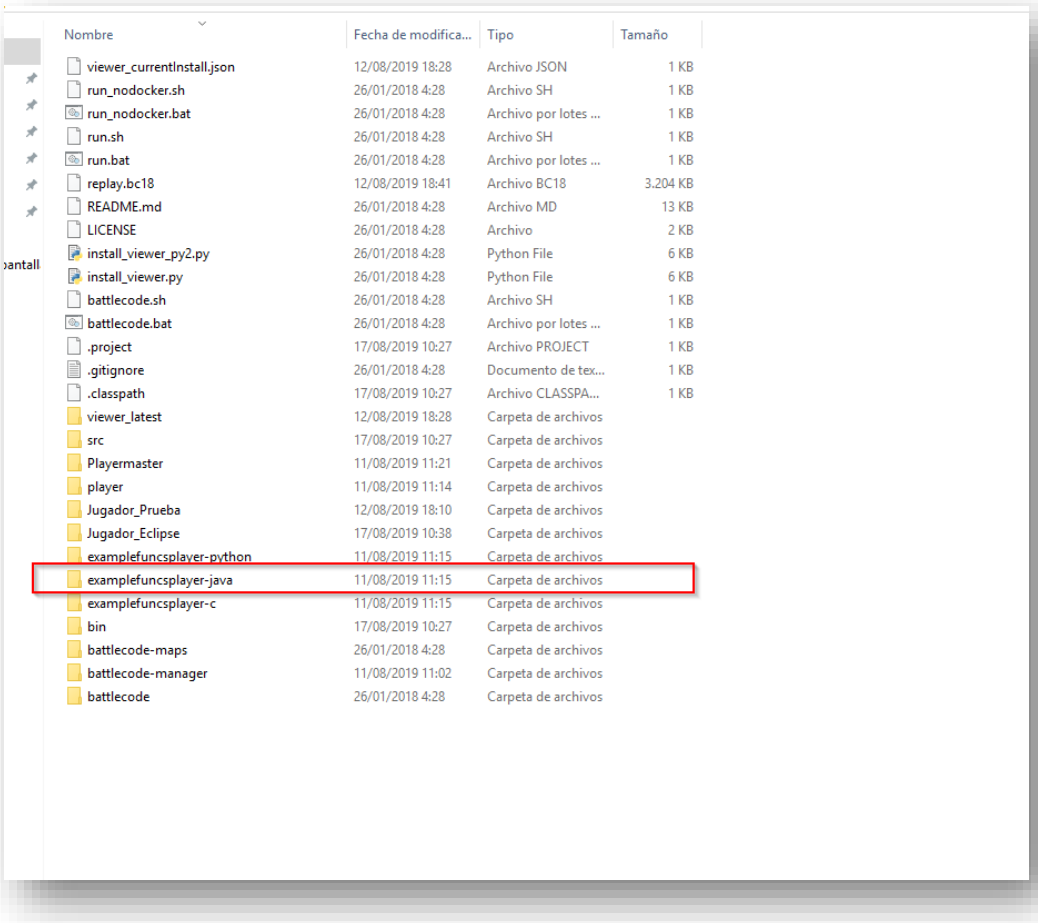

Figura 36: Eclipse (11): Carpeta examplefuncsplayer-java

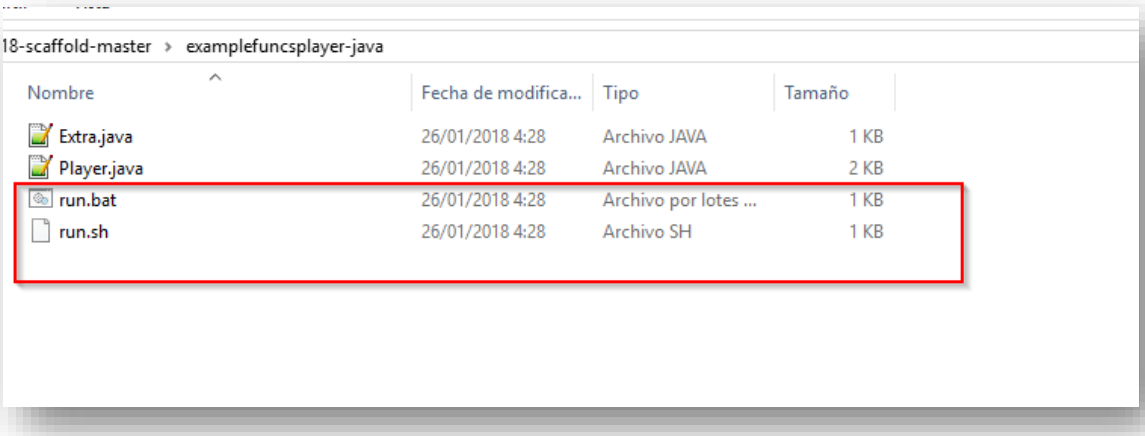

Figura 37: Eclipse (12): Copia de fichero

Los archivos copiados se pegarán en la carpeta que se ha creado Jugador_Eclipse.

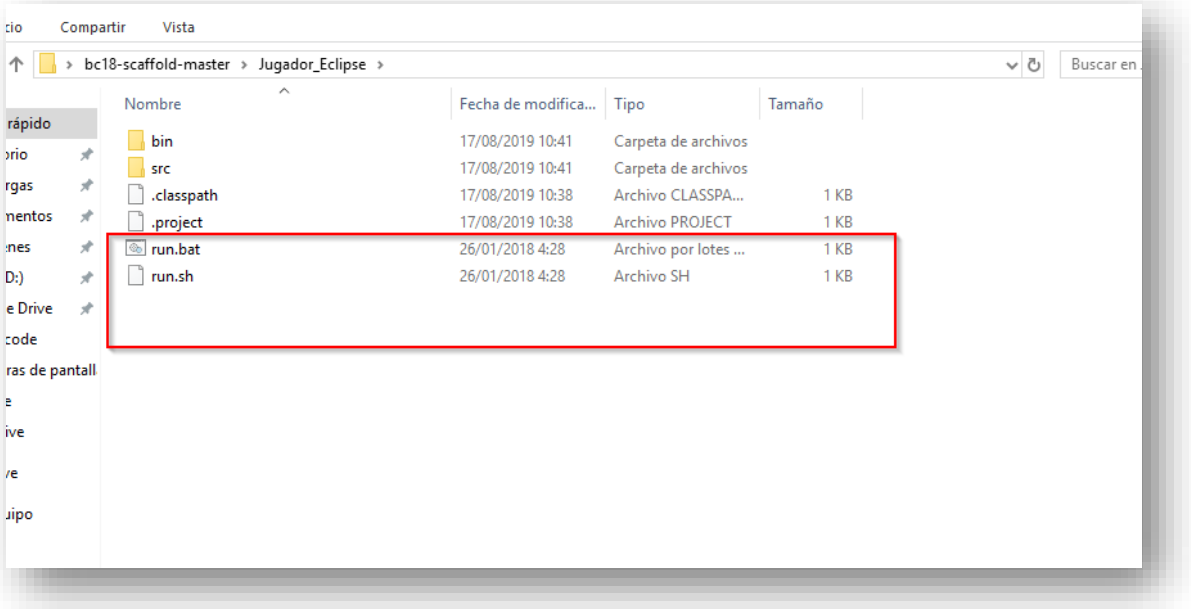

Figura 38: Eclipse (13): Pegar fichero carpeta JUGADOR_ECLIPSE

Una vez copiado los dos archivos run, se abrirá el run.sh con un editor de texto y sustituirá todo su contenido por este:

```
#!/bin/sh
# This file should build and run your code.
# It will run if you're in nodocker mode on Mac or Linux,
# or if you're running in docker.

# Compile our code.
#echo javac $(find . -name '*.java') -classpath ../battlecode/java
#javac $(find . -name '*.java') -classpath ../battlecode/java

# Run our code.
echo java -classpath .:bin Player
java -classpath .:bin Player
```

Una vez sustituido el contenido se guardará y con esto se termina la preparación del Eclipse.

4.3 Debugging

Los errores que se producen en la ejecución del programa se muestran en la consola del navegador web cuando se ejecuta Battlecode. Para poder ver los errores de esta manera se tendrá que haber ejecutado los pasos anteriores.

Para ver mientras se programa los errores que se va cometiendo se debe tener abierto Battlecode y Eclipse.

Siempre antes de ejecutar una partida desde el navegador web se tendrá que guardar los cambios en eclipse, seleccionando el botón de la Figura 39.

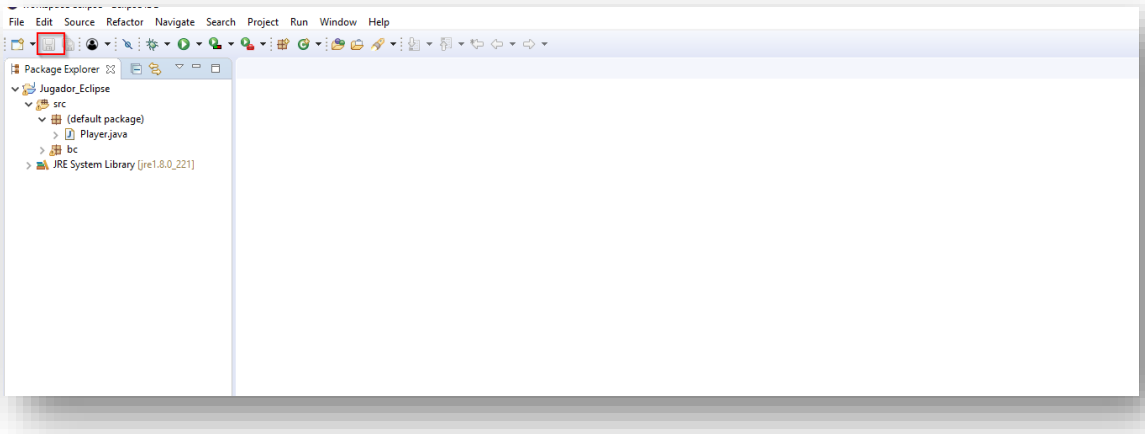

Figura 39: Eclipse (14): Guardar eclipse

Cuando ya se haya guardado los cambios, seleccionará nuestro robot y se le dará Run Game

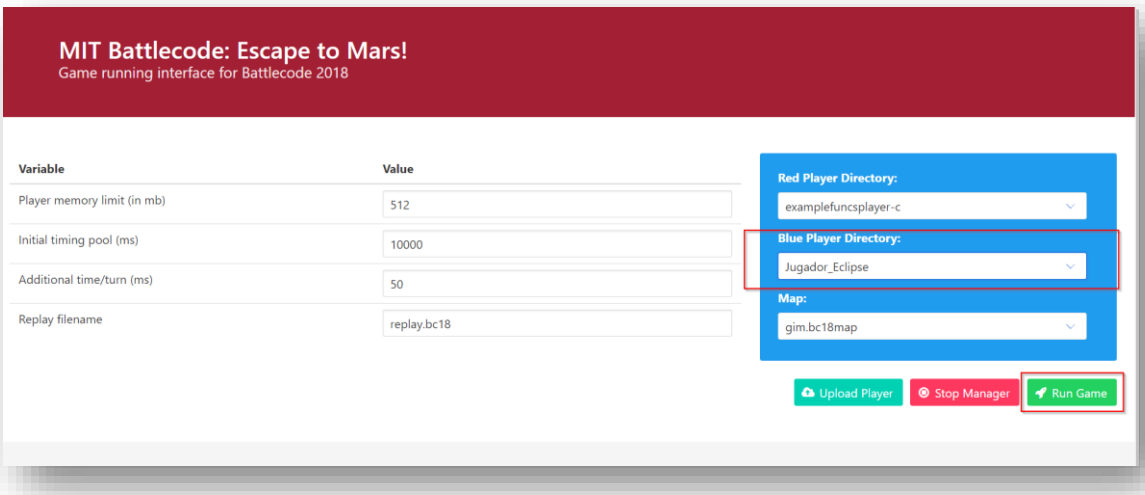

Figura 40: Eclipse (15): Navegador Battlecode

Y en la Figura 41, en el recuadro rojo es donde está la consola, todo lo que se quiera mostrar por consola o los errores que dé a ejecutar el robot saldrán en dicho recuadro.

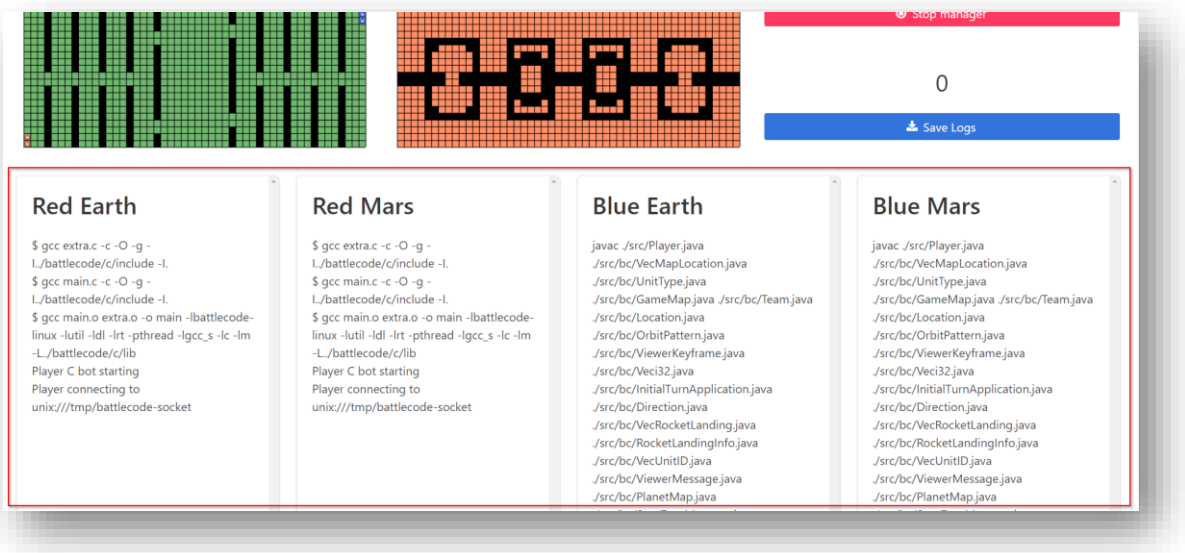

Figura 41: Eclipse (16): Consola en Battlecode

4.4 Descripción de un Jugador

Según la estrategia que se quiera elegir, un jugador puede constar de diferentes robots o estructuras.

Hay diversos tipos de robots y estructuras, según su utilización y sus características.

Los robots son los siguientes.

Los Workers son robots que sirven para recolectar minerales (Karbonite), crear y reparar estructuras, para crear una estructura se hace en dos pasos, primero se crea el plano y después lo construye. Estos robots se crean duplicándose entre sí.

Los siguientes robots se crean mediante una estructura llamada Factory, estos robots sirven para atacar a los enemigos o curar a los aliados.

Los Knights son robot de cuerpo a cuerpo, tienen más aguante que los demás robots, son utilizados como tanques, es decir, los que se ponen en primera línea de combate.

Los Rangers son robots que disparan a distancia y atacan a un solo enemigo. Estos, juntos a los Mages, que se utilizaban más para hacer daño en área, se ponen detrás de los Knights, porque tienen menos vida que estos.

Y por último están los Healers, que son los robots de apoyo, que les va dando vida a sus aliados para aguantar más en el combate.

A continuación, se describen las estructuras, hay dos tipos:

Los Rockets que son los cohetes para viajar a otro planeta. Se utilizan en los combates porque en el comienzo de la batalla se empieza en la Tierra, pero si en la Tierra no se decide quien ha ganado, en el turno 750 de la partida cae un meteorito y la destruye, si los jugadores no se han llevado sus robots a Marte para seguir la batalla pierden la partida.

La otra estructura es la Factory que es para crear los robots que van a luchar contra los enemigos.

4.5 Creación de un jugador

En este apartado se explicará cómo crear un equipo mediante el lenguaje de programación Java. Aparte de los métodos que se van a ver en este apartado existen muchos más en la api del Battlecode (https://s3.amazonaws.com/battlecode-2018/api/java/index.html).

Antes de empezar a programar se recomienda mirar las especificaciones de cada estructura y robots en la dirección (https://s3.amazonaws.com/battlecode-2018/specs/battlecode-specs-2018.html).

Aquí se va a exponer una forma de crear un robot, hay muchas formas de crearlo según la lógica que se vaya a utilizar, pero la clase principal que lleva el método MAIN, siempre tiene que llamarse Player.

En cada explicación de cada clase de este apartado se encontrará un diagrama que indica el nombre de la clase, los atributos y las funciones de la clase como se puede ver en este ejemplo.

Nombre de la clase
Atributos
Funciones()

En el Anexo se puede ver el código de todas las clases explicadas en este apartado.

Las diferentes clases son:

1. *Clase Player.*

Player
stactic Utilidades utilidades static GameController GC
public static void MAIN(String[] args)

Esta es la clase principal desde donde se llamará a las otras clases.

Utilizando la clase Gamecontroller, que es una clase de la api Battlecode, se utilizará el método myUnits(), que devuelve un array de las unidades que hay en ese momento en la partida.

Se recorrerá ese array de unidades y según el tipo de unidad, se llamará al método correspondiente a dicha unidad.

2. *Clase Utilidades*

Utilidades
static Gamecontroller gc static Direction[] directions static VecUnit unidadesEnemigas static MapLocation resultado static final PlanetMap planetaTierra static final int anchoMapaTierra static final int altoMapaTierra static final int nodoMaxTierra static HashMap<String, Integer> passableTierra static HashMap<Integer, String> passableTierra2 static int adyTierra[][] static HashMap<String, Long> karboMapTierra static Team miEquipo static Team equipoEnemigo static int countFactory static int countHealer static int countKnight static int countMage static int countRanger static int countRocket static int countWorker

```
public Utilidades()
public static boolean checkPassable(MapLocation test)
public static void initialize()
public static int[][] relacionNodeTierra()
public static Direction bfs(MapLocation mapini, MapLocation mapfin)
public static MapLocation distance(MapLocation first, HashMap<String, Long> map)
static public void contarUnidades(VecUnit units)
public static MapLocation destino(Unit unit)
public static void mover(Direction direction, Unit unit)
public static void atacar(Unit unit)
public static void producirRobot(Unit unit, UnitType unitType)
```

La clase Utilidades que se ha creado para este ejemplo de creación de robot, tendrá los métodos comunes que se utilizarán en las demás clases: Player, Factory, Ranger, Worker.

A continuación, se explicará los distintos métodos que se tendrá en esta clase utilidades.

2.1 CHECKPASSABLE

Este método nos devuelve un boolean que indica si la localización que le se le pasará es pasable o no, es decir, si los robots van a poder pasar por ese punto.

2.2 INITIALIZE

Este método no devuelve nada solo sirve para inicializar tres HashMap. El primer HashMap llamado passableTierra, guarda en la clave las coordenadas y en el valor un número que, representa un nodo o un cuadrado del mapa. El segundo hashmap llamado passableTierra2 en su clave se guardará el nodo y en el valor las coordenadas. Y en el tercer hashmap, llamado karboMapTierra, se guardan las coordenadas y la cantidad de minerales que hay en el juego (Karbonite) que hay en esa ubicación.

2.3 RELACIONNODETIERRA

Este método se utilizará para relacionar los nodos entre sí y poder luego seleccionar el camino más corto.

2.4 BFS

Este método devuelve el camino más corto que hay entre ellos.

2.5 DISTANCE

Mediante las localizaciones que se le indican, devolverá la distancia entre ellos.

2.6 CONTARUNIDADES

Este método ayudará a tener un control de nuestras unidades, contando las unidades que hay en cada momento.

2.7 DESTINO

Este método nos informará y guardará en un array los enemigos que hay alrededor del destino.

2.8 MOVER

Introduciéndole la dirección donde se va a mover a mover y la unidad que es, nos dirá si va a poder moverse a esa dirección o no.

2.9 ATACAR

Con este método se podrá saber si las unidades que tienen alrededor son enemigas y si son enemigas se podrá atacar. Aparte de poder atacarlas, se guardará la posición de las unidades enemigas en el array llamado resultado, para que cualquier aliado pueda ir atacar esa posición.

2.10 PRODUCIRROBOT

Este método se utilizará para ver si se puede construir un robot.

3. Clase Factory

Factory
static Unit unit static GameController gc
public static void ejecutar(GameController gc, Unit unit)

Esta clase se encarga de crear robots.

Para crear un robot se utilizará el método de la clase Utilidades producirRobot(), que lo introduce en una estructura de guarnición. Y una vez que está listo para salir de la guarnición mediante el método unload() se creará y ya podrá combatir.

4. *Clase Worker*

Worker
static Unit workerBuid static int limiteFactory static int idFactoryBluePrint
public static void ejecutar(GameController gc, Unit unit)

En la primera ronda se va a duplicar un worker. Se duplican los workers para poder recolectar más recursos más rápido y poder ir creando las Factorías mientras otros workers recolectan minerales (Karbonite). Los workers se crean duplicándose entre ellos.

Los Workers también pueden crear factoría o un cohete, la construcción se hacen en dos pasos, uno crear un plano con el método blueprint(), y dos una vez creado el plano ya se podrá construir la factoría o el cohete con el método build().

Otra acción que realizan los workers es recolectar los minerales que en el Battlecode se llama Karbonite, para coger los minerales se utiliza el método harvest().

5. *Clase Ranger*

Ranger
public static void ejecutar(GameController gc, Unit unit)

En el Battlecode hay más de un tipo robot que pueden atacar, en este proyecto como ejemplo se ha creado un tipo Ranger.

Cuando en nuestras unidades hayan 10 Rangers, estos podrán ir en busca de los enemigos y atacar.

Conclusiones

Battlecode es un juego para iniciarse en el mundo de la programación, no es complicado el uso de la api para la implementación de los equipos y además resulta entretenido compitiendo con otros jugadores. La parte aburrida es preparar todo para poder empezar a programar, pero eso se hace solo una vez y ya tienes todo el tiempo para pensar tus estrategias y divertirte programando.

Antes de empezar con las instalaciones se deben de leer muy bien como implementar cada programa que se va a utilizar en este proyecto de sus páginas oficiales, para evitar durante la instalación algún error inesperado.

Además, se recomienda que se lean muy detenidamente las especificaciones del juego y que se tengan a mano la api del Battlecode antes de empezar a programar.

Para ampliar este trabajo fin de grado en un futuro, se puede hacer la implementación de mapas nuevos he introducirlos al juego para tener más opciones personalizados al gusto del jugador.

El juego cumple con su misión de ayudar a los alumnos a afianzar los conceptos aprendidos sobre la programación prácticamente sin darse cuenta por el hecho de estar haciendo algo que no es aburrido para ellos ya que el final es jugar con las estrategias que hayas sido capaz de diseñar y llevar a la práctica.

Personalmente lo recomiendo como herramienta para el aprendizaje y la adquisición de habilidades y conocimientos de programación.

Anexos.

1- Docker instalación

1.1 Instalación del Docker

Docker se puede instalar en Mac, Linux y Windows. La instalación se ha probado en Windows 10 Home, Windows 10 Pro y Linux y en esos sistemas operativos ha funcionado correctamente. En este documento se verá una de las dos instalaciones que se pueden hacer en Windows 10.

Según la versión de Windows que se tenga, podrá soportar las dos instalaciones o solo una. Por ejemplo, la versión de Windows 10 Home solo podrá soportar la instalación de Docker de la forma que se va a explicar en este documento y la versión de Windows 10 Pro puede soportar las dos versiones de instalación que hay para Docker.

En este apartado se explicará la instalación del Docker que funciona en cualquier versión de Windows.

A la hora de hacer este documento la versión del Docker es 18.03.0-ce, el proceso de la instalación y la versión del Docker puede cambiar según la fecha de la instalación.

Se recomienda que, si se tiene la versión de Windows 10 Pro o Profesional, se instale el Docker correspondiente a su versión (https://www.docker.com/products/docker-desktop)

A continuación, se verá los pasos a seguir para la instalación del Docker.

A. Preparación de Archivos.

- Se deberá tener la virtualización activada como se muestra en la Figura 42. Si la virtualización está deshabilitada se tendrá que entrar a la BIOS del ordenador y habilitarla.

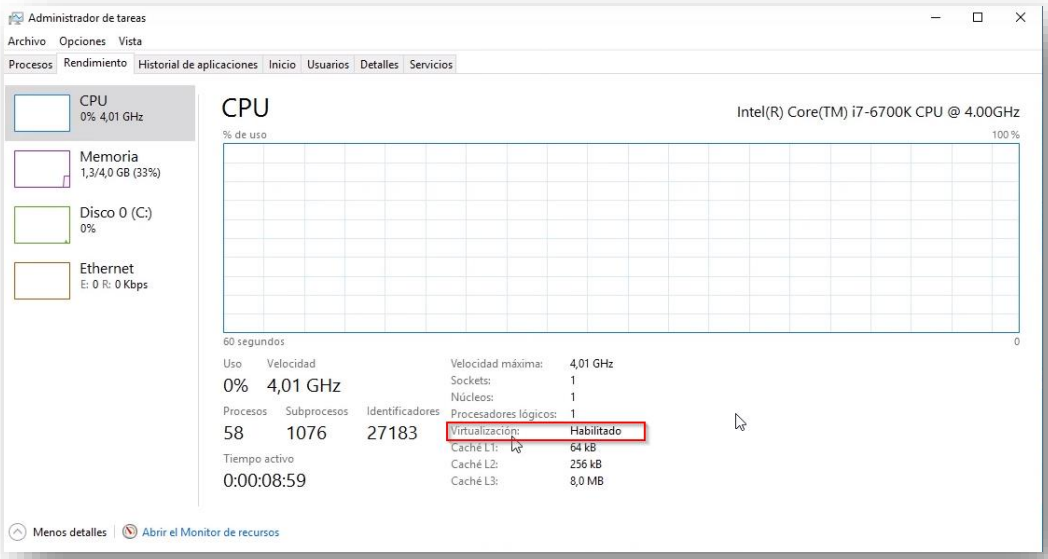

Figura 42: Virtualización:Administrador de tareas

- En el siguiente enlace (https://docs.docker.com/toolbox/toolbox_install_windows/) nos llevará a la documentación del Docker y se pinchará en el recuadro rojo que pone Toolbox Release como muestra la Figura 43.

Se deberá tener instalado Python 3.6.4 de 64 bits y se podrá descargar de este enlace (https://www.python.org/ftp/python/3.6.4/python-3.6.4-amd64.exe)

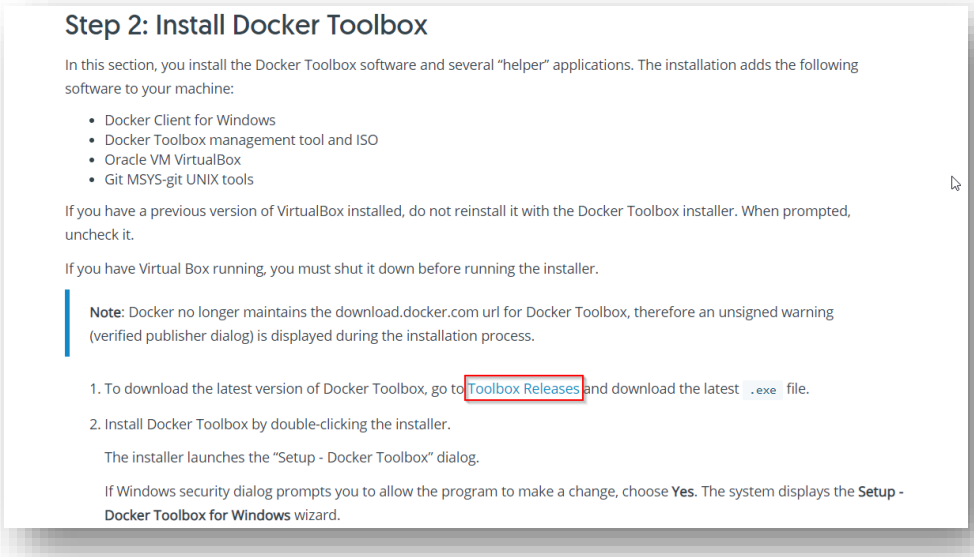

Figura 43: Documentación Docker

- El enlace anterior nos direccionará a esta dirección (https://github.com/docker/toolbox/releases). Aquí se tendrá que buscar en las versiones más reciente el archivo con extensión .exe. En el supuesto caso que en la versión más reciente no salga ese archivo se irá buscando en las versiones anteriores. En la fecha que se hizo este documento se descargará esta versión DockerToolbox-18.03.0-CE.exe. Se pinchará en el recuadro rojo que se observa en la Figura 44 para descargar el archivo.
-

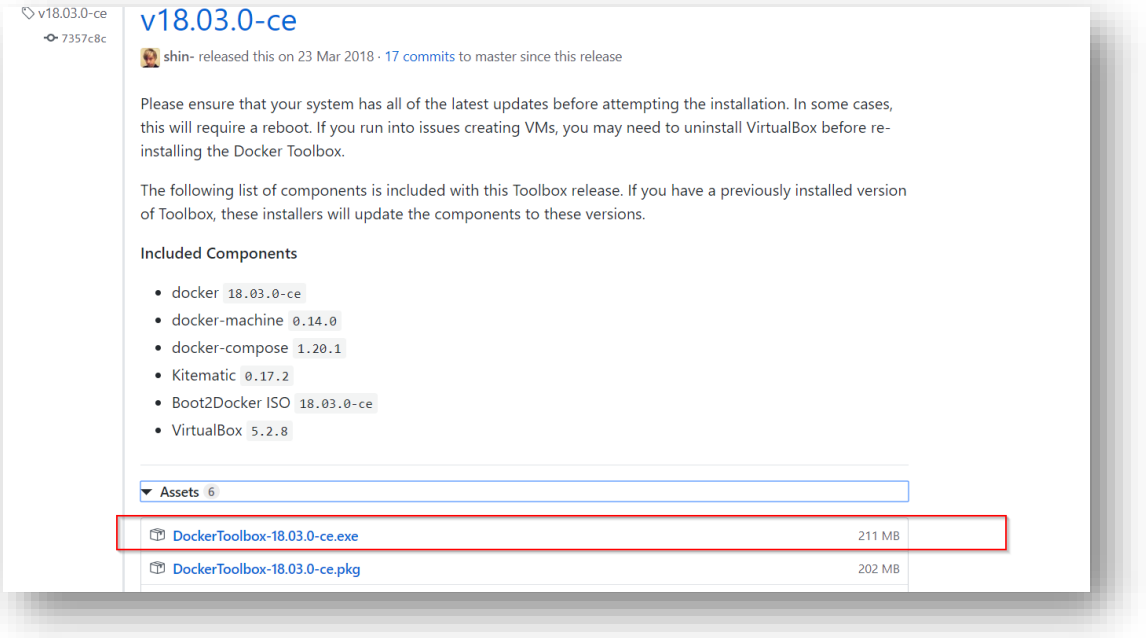

Figura 44: Descarga de Docker

B. Instalación

Una vez realizado los pasos anteriores necesarios para la instalación del Docker, se procederá de la siguiente manera:

Se ejecutará el archivo anterior descargado.

En las siguientes imágenes se ve todo el proceso de instalación. Para ir saltando de ventana en ventana se tendrá que ir dando al botón NEXT hasta llegar a la ventana de instalación que se tendrá que dar en este caso al botón INSTALL de la Figura 49.

Para un buen funcionamiento del programa se recomienda que se seleccione las opciones marcadas como muestran las siguientes figuras.

Figura 45: Instalación Docker (1)

En la Figura 46 se seleccionará la carpeta donde se quiere instalar Docker.

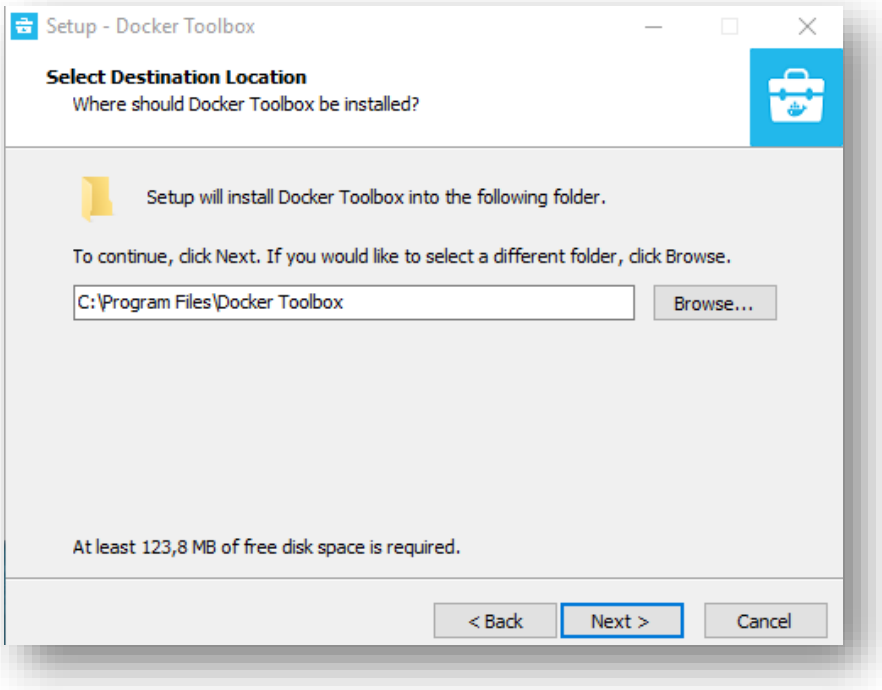

Figura 46: Instalación Docker (2)

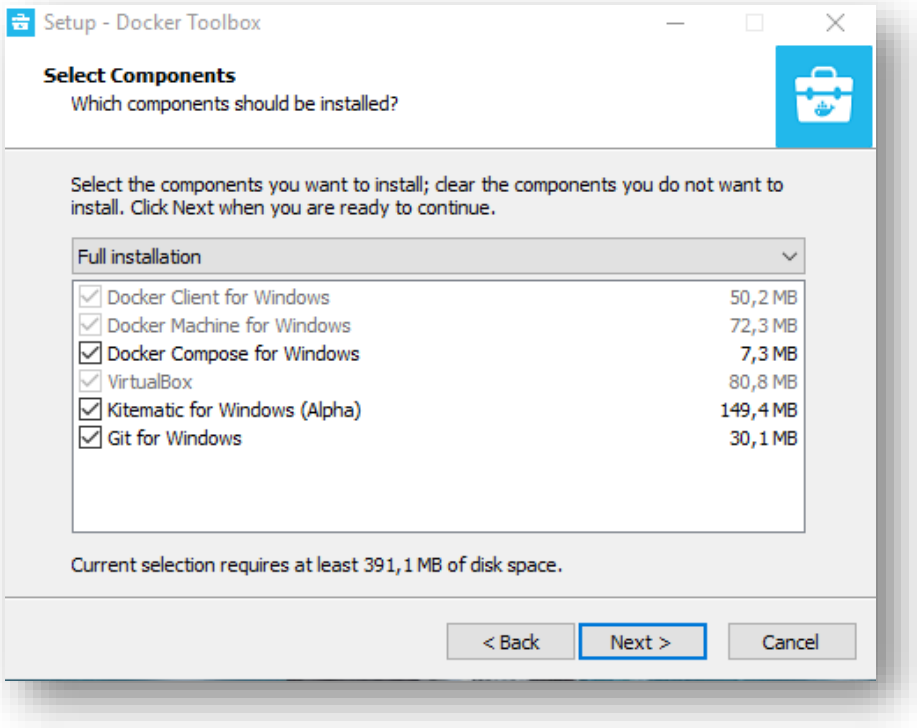

Figura 47: Instalación Docker (3)

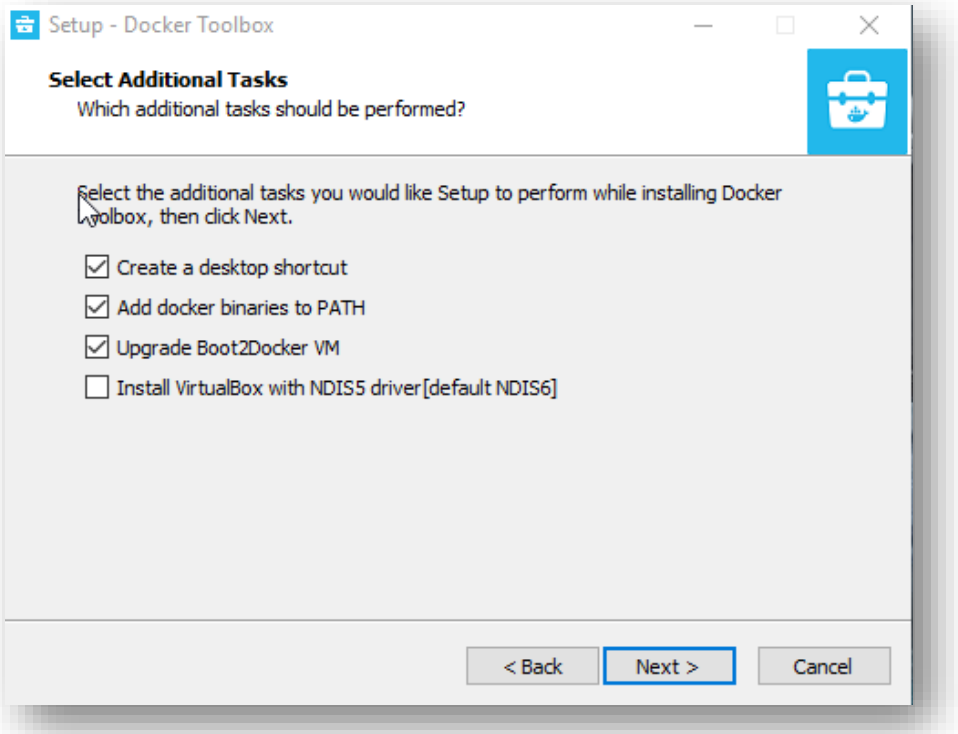

Figura 48: Instalación Docker (4)

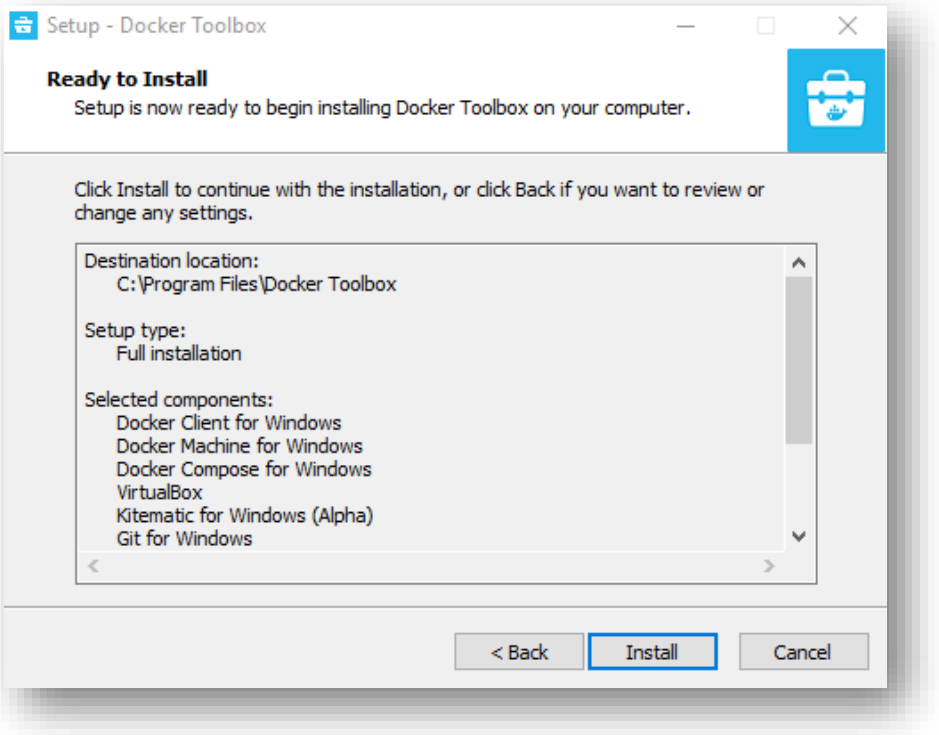

Figura 49: Instalación Docker (5)

En la Figura 50, se verá el progreso de la instalación.

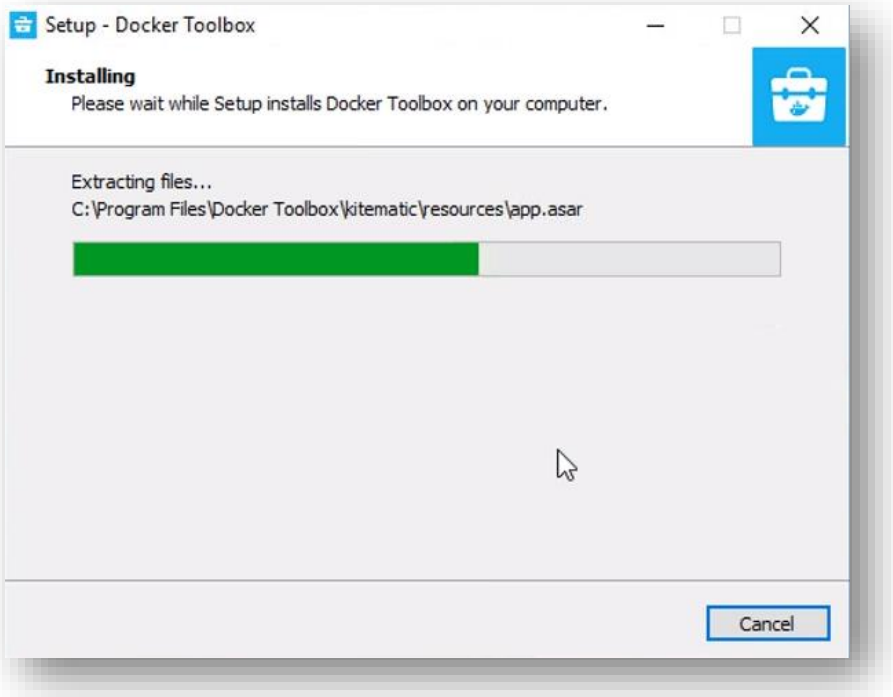

Figura 50: Instalación Docker (6)

Cuando termine la instalación nos saldrá la ventana de la Figura 51 y se clicará en FINISH, acabando así la instalación del programa.

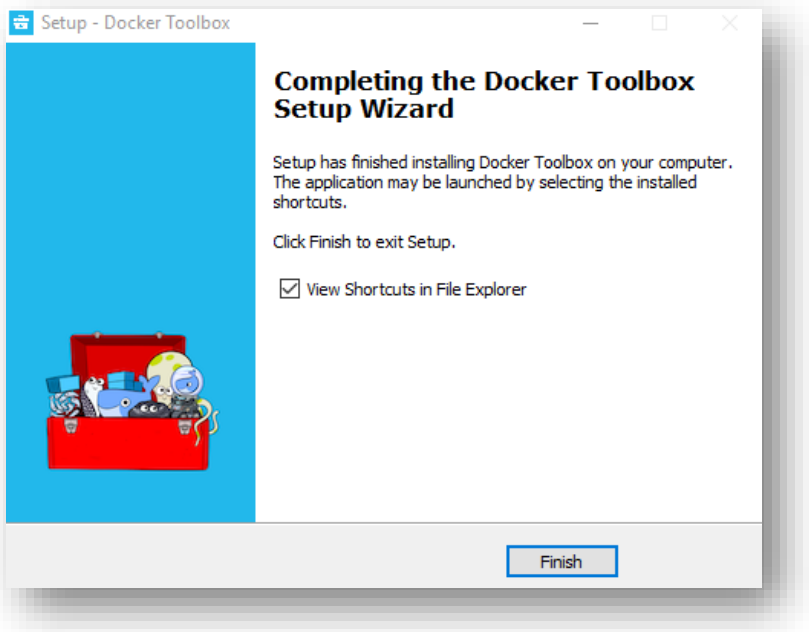

Figura 51: Instalación Docker (7)

C. Comprobación de la instalación

- En nuestro escritorio se habrán creado tres accesos directos después de la instalación (Figura 52).

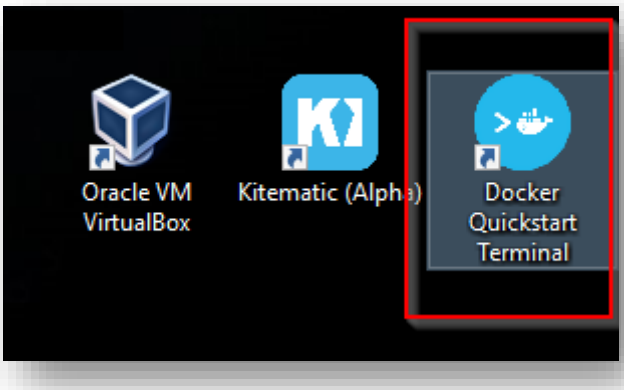

Figura 52: Instalación Docker (8): Accesos directos

- Se ejecutará Docker Quickstart Terminal. Si tras ejecutarlo, sale la ventana de la Figura 53, se tendrá que ir al apartado *1.2 Solución de problemas* y

seguir los pasos que se indica. Pero en cambio, si sale la Figura 54 es que la instalación avanza correctamente, habrá que esperar a que se configure automáticamente Docker para un buen funcionamiento, mientras se está configurando puede salir varias ventanas y tendréis que darle a SI, para que pueda seguir la configuración.

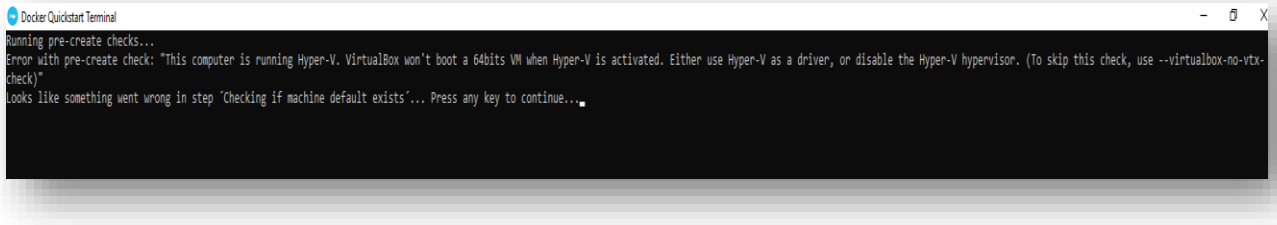

Figura 53: Instalación Docker (9): Error

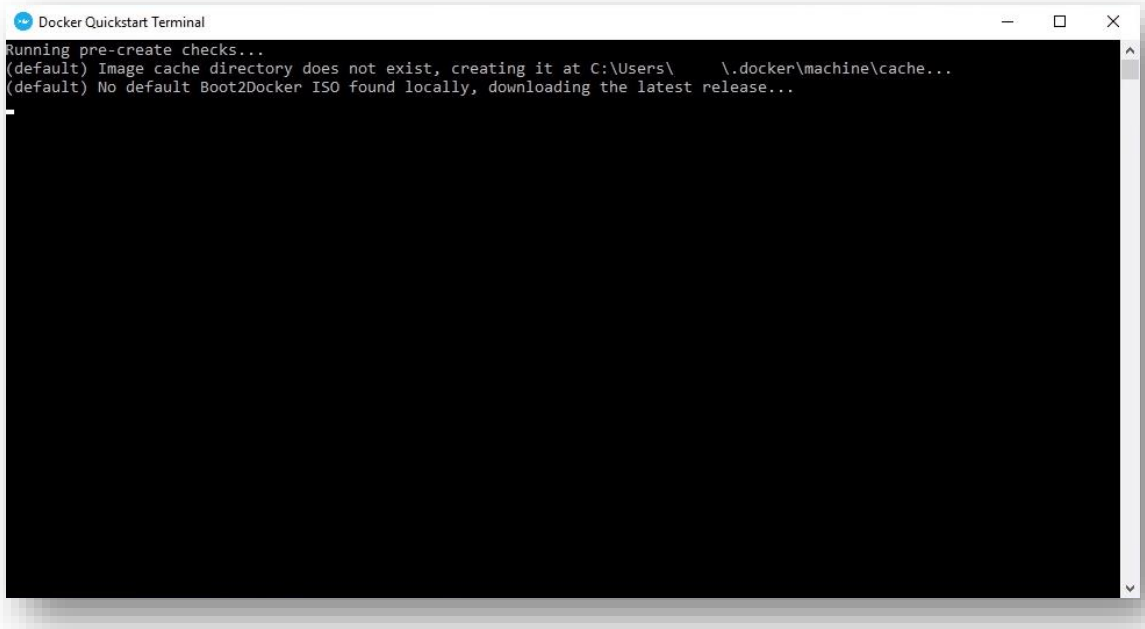

Figura 54: Instalación Docker (10): Ejecutando Docker

Cuando haya terminado la configuración saldrá una ventana como la Figura 55.

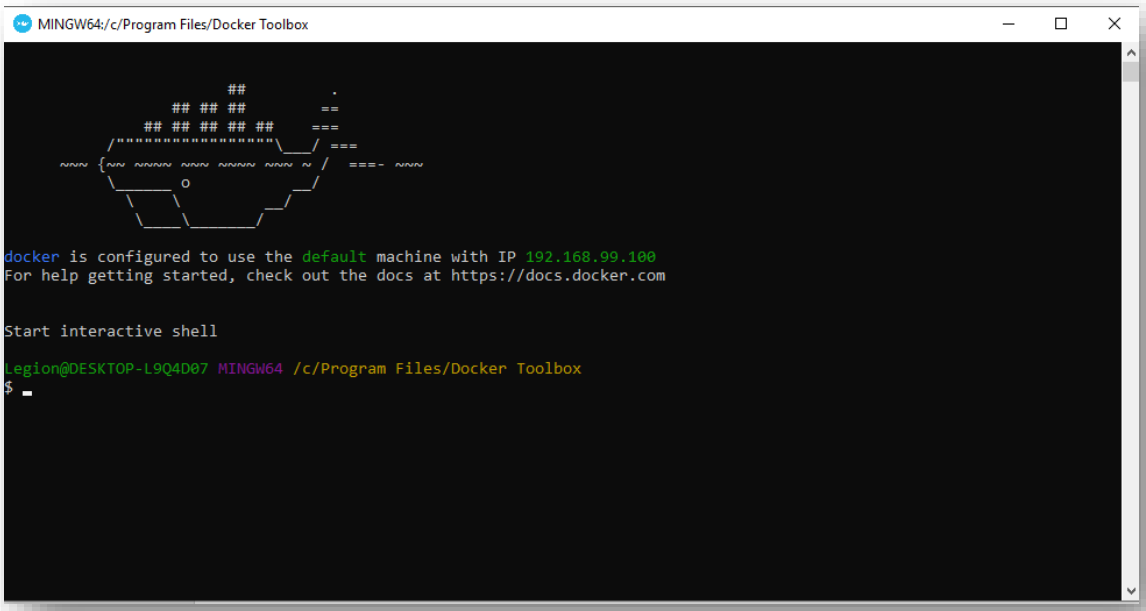

Figura 55: Instalación Docker (11): finalización de instalación Docker

Para terminar con la instalación y comprobar que ha ido todo correctamente en la ventana de la Figura 15 se escribirá el comando: docker --version y tendrá que devolver la versión del Docker como se muestra en la Figura 56 que pone

“ Docker version 18.03.0-ce, build 0520e24302” .

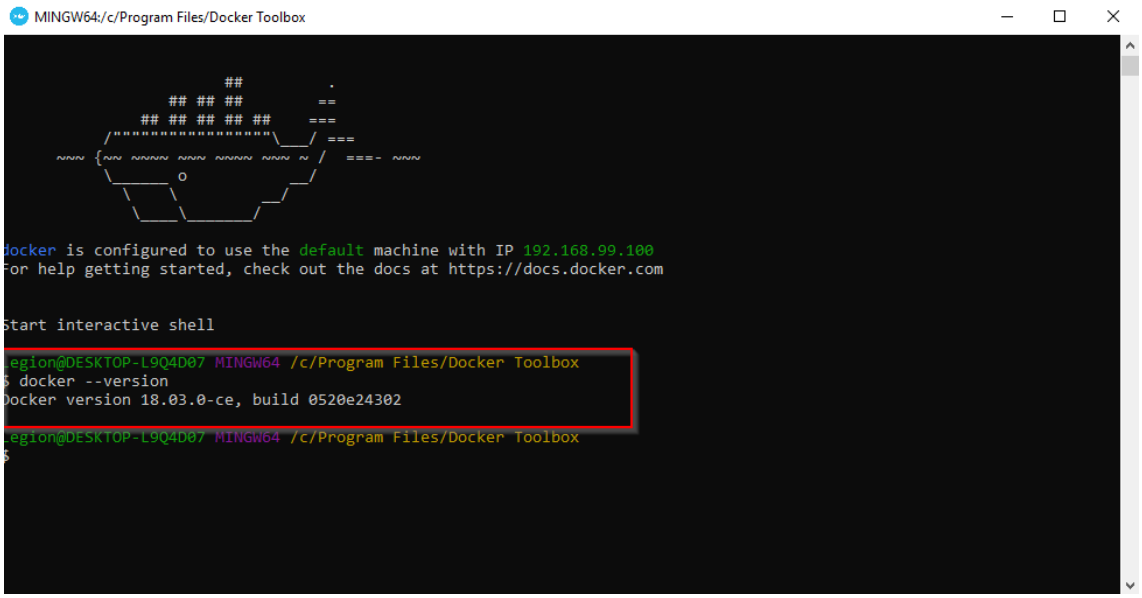

Figura 56: Instalación Docker (12): Ver versión Docker

Una vez finalizado la instalación de Docker, se podrá pasar a configurar Battlecode 2018.

1.2 Solución de problemas

Para solventar el problema que nos ha dado en el apartado 3 Comprobación de la instalación, se tendrá que desactivar el HYPER-V.

Para desactivarlo se seguirá los siguientes pasos:

A. Primero se seleccionará en el botón de inicio de Windows, abajo a la izquierda de la barra de herramientas y se escribirá “panel de control” como muestra la Figura 57.

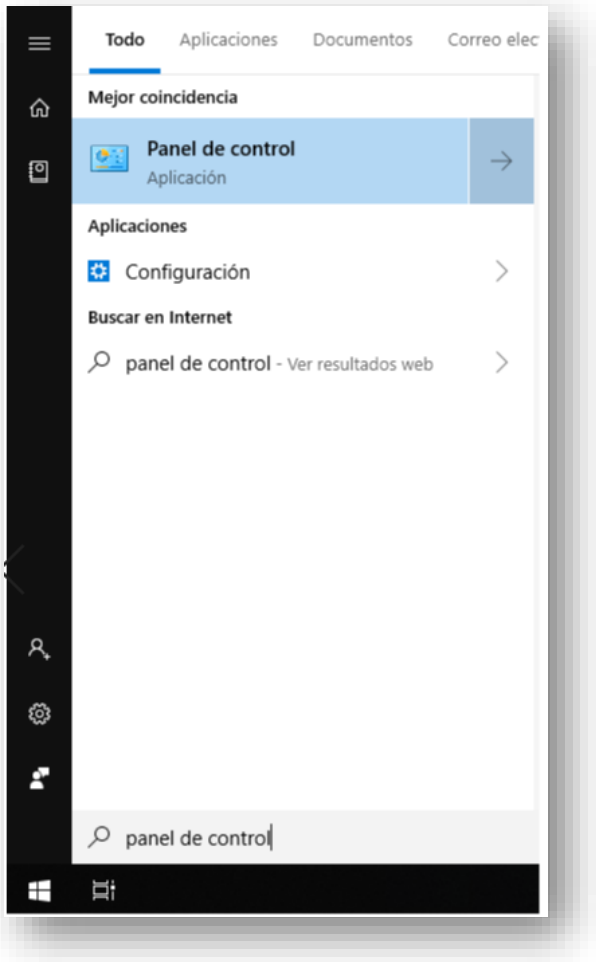

Figura 57:Solución de problemas Docker (1)

B. En el apartado de programas se seleccionará en la opción que pone: Desinstalar un programa (Figura 58).

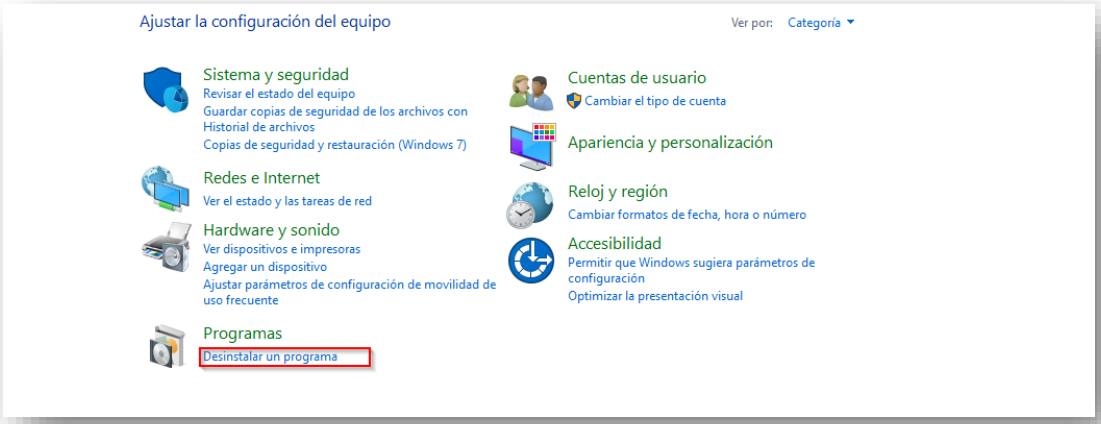

Figura 58:Solución de problemas Docker (2): Panel de control

C. Como se muestra en la Figura 59 se irá a la opción seleccionada que está en el recuadro en rojo: Activar o desactivar las características de Windows.

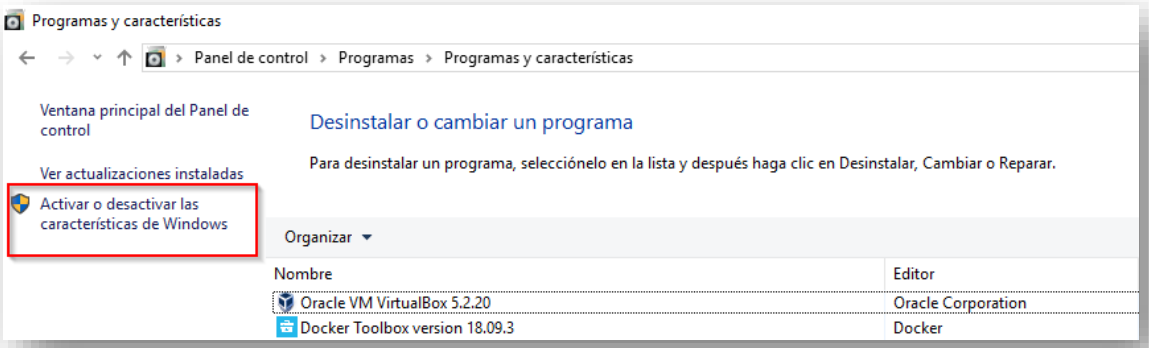

Figura 59:Solución de problemas Docker (3). Desinstalar o cambiar un programa

D. Cuando muestre la lista de opciones, se buscará la opción de Hiper-V y será desmarcada. Se tiene que quedar como muestra la Figura 60, y a continuación se aceptará.

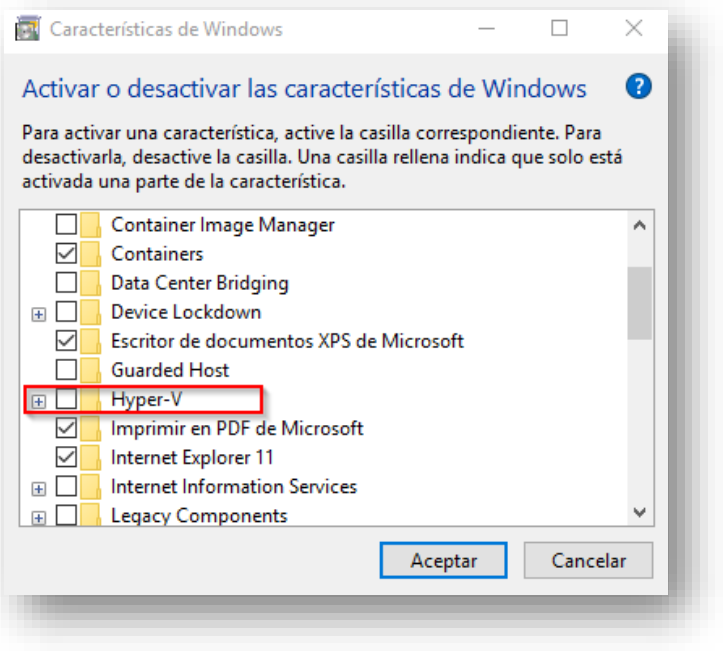

Figura 60:Solución de problemas Docker (4): Activar o desactivar características

E. Tras haber pulsado aceptar en el paso anterior, aparecerá la ventana de progreso de aplicación de cambios, a su finalización se mostrará otra ventana (Figura 62) donde se seleccionará la opción de reiniciar ahora.

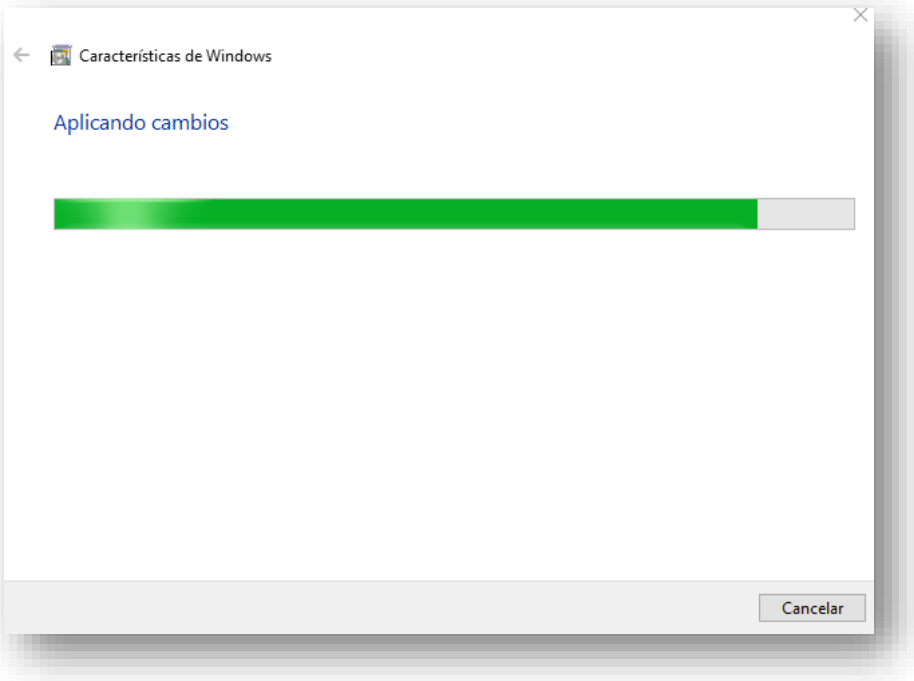

Figura 61:Solución de problemas Docker (5)

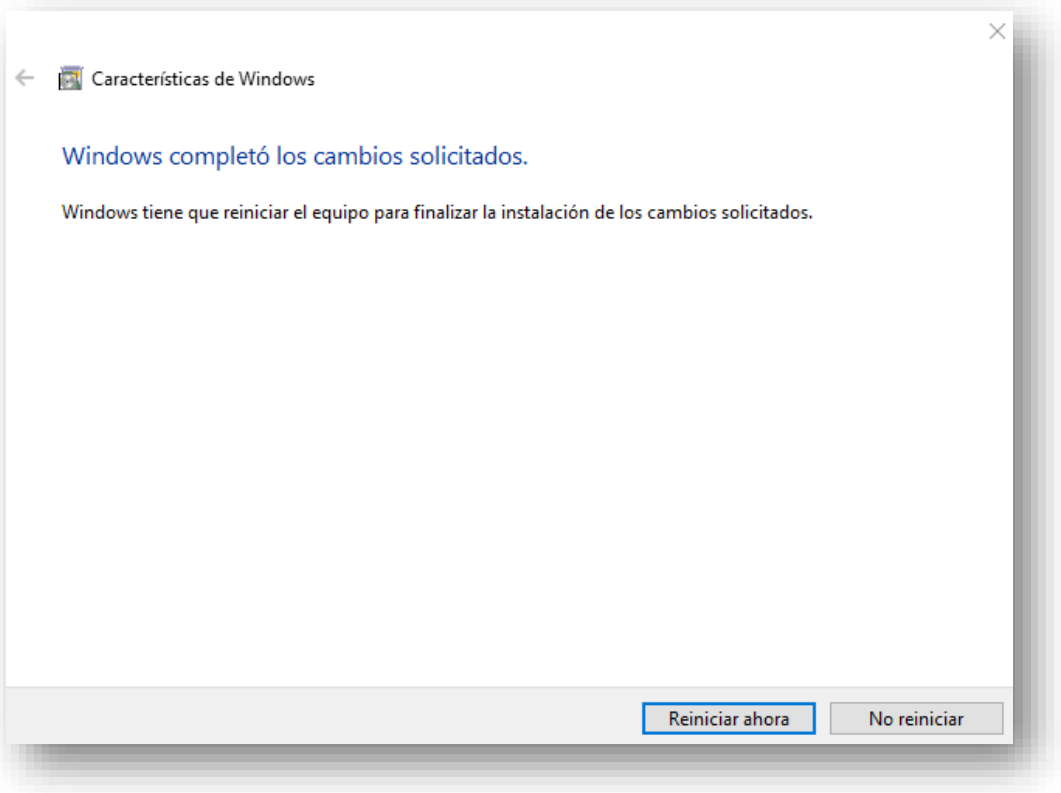

Figura 62:Solución de problemas Docker (6)

Cuando se haya reiniciado y configurado se podrá proseguir con los pasos de la instalación.

2 - Código Jugador Battlecode 2018.

La estructura final que debe de tener eclipse con todos los elementos explicados anteriormente se muestra en la Figura 63.

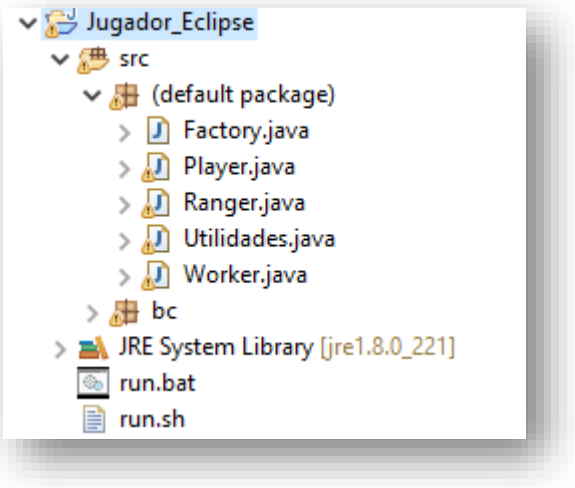

Figura 63:Estructura Robot Eclipse

Se va a mostrar los códigos de las clases que compone el robot que se ha creado para este trabajo, se ha utilizado como referencia: (https://github.com/VoidMercy/battlecode-2018), y para calcular el camino más corto el que hay entre dos robot (bfs) (https://github.com/jariasf/Online-Judges-Solutions/tree/master/Algorithms/JAVA/Graph%20Theory).

A. Clase Player

Código:

```
import bc.*;

public class Player {

	static Utilidades utilidades = new Utilidades();
	static GameController gc = utilidades.gc;

	public static void main(String[] args) {

		VecUnit units;
		Unit unit;
		utilidades.initialize();

		try {
			while (true) {

	// System.out.println("[Player] Current round: " + gc.round());
```

```
                                units = gc.myUnits();

                                utilidades.contarUnidades(units);

                                long size = units.size();
                                for (int i = 0; i < size; i++) {

                                        unit = units.get(i);

                                        if (unit.health() > 0) {

                                                switch (unit.unitType()) {
                                                case Factory:
                                                        try {
                                                                Factory.ejecutar(gc, unit);
                                                        } catch (Exception e) {
                                                                e.printStackTrace();
                                                        System.out.println("Factory ded");
                                                        }
                                                        break;

                                                case Ranger:
                                                        try {
                                                                Ranger.ejecutar(gc, unit);
                                                        } catch (Exception e) {
                                                                e.printStackTrace();
                                                        System.out.println("Range ded");
                                                        }
                                                        break;

                                                case Worker:
                                                        try {
                                                        Worker.ejecutar(gc, unit)
                                                        } catch (Exception e) {
                                                                e.printStackTrace();
                                                        System.out.println("worker ded");
                                                        }
                                                        break;

                                                }

                                        }


                                }
                                        gc.nextTurn();

                        }

                } catch (Exception e) {
                        e.printStackTrace();
                        System.out.println("Fail player");
                }
        }
}
```

B. Clase Utilidades

Código.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import bc.*;
```

```java
public class Utilidades {

	// Connect to the manager, starting the game
	static GameController gc = new GameController();

	static Direction[] directions = Direction.values();

	static VecUnit unidadesEnemigas;
	static MapLocation resultado;

	// Variable para Tierra
	static final PlanetMap planetaTierra = gc.startingMap(Planet.Earth);
	static final int anchoMapaTierra = (int) planetaTierra.getWidth();
	static final int altoMapaTierra = (int) planetaTierra.getHeight();
	static final int nodoMaxTierra = anchoMapaTierra * altoMapaTierra;
	static HashMap<String, Integer> passableTierra = new HashMap<String, Integer>();
	static HashMap<Integer, String> passableTierra2 = new HashMap<Integer, String>();
	static int adyTierra[][];
	static HashMap<String, Long> karboMapTierra = new HashMap<String, Long>();

	// Variable para Marte
	static final PlanetMap planetaMarte = gc.startingMap(Planet.Mars);
	static final int anchoMapaMArte = (int) planetaMarte.getWidth();
	static final int altoMapaMarte = (int) planetaMarte.getHeight();
	static final int nodoMaxMarte = anchoMapaTierra * altoMapaTierra;
	static HashMap<String, Integer> passableMarte = new HashMap<String, Integer>();
	static HashMap<Integer, String> passableMarte2 = new HashMap<Integer, String>();
	static int adyMarte[][];
	static HashMap<String, Long> karboMapMarte = new HashMap<String, Long>();

	// Equipo
	static Team miEquipo = gc.team();
	static Team equipoEnemigo;

	// Contador de unidades
	static int countFactory = 0;
	static int countHealer = 0;
	static int countKnight = 0;
	static int countMage = 0;
	static int countRanger = 0;
	static int countRocket = 0;
	static int countWorker = 0;

	public Utilidades() {

	}

	public static boolean checkPassable(MapLocation test) {

		PlanetMap planeta;
		if (test.getPlanet() == planetaTierra.getPlanet()) {

			planeta = planetaTierra;

		} else
			planeta = planetaMarte;

		/*
		 * Si existe una unidad en la casilla test, está Libre es falso y si no existe
		 * lanza la excepcion y la casilla es true
		 */
		boolean isFree;
		try {
			gc.senseUnitAtLocation(test);
			isFree = false;
		} catch (Exception e) {
			isFree = true;
		}

		/*
		 * Comprobar que la casilla no esté ocupada por estructuras y que sea pasable
y
```

```
                 * lanza una excepción si sale del mapa. además de que la casilla tiene que
estar
                 * libre.
                 */
                try {
                        isFree = planeta.isPassableTerrainAt(test) == 1 && isFree;

                } catch (Exception e) {
                        isFree = false;
                }
                return isFree;
        }

        public static void initialize() {

                if (Utilidades.miEquipo == Team.Red) {
                        Utilidades.equipoEnemigo = Team.Blue;
                } else {
                        Utilidades.equipoEnemigo = Team.Red;
                }

                String string = null;
                int contador = 0;
                for (int i = 0; i < altoMapaMarte; i++) {
                        for (int a = 0; a < anchoMapaTierra; a++) {
                        MapLocation temp = new MapLocation(planetaTierra.getPlanet(), a, i);
                                if (planetaTierra.isPassableTerrainAt(temp) == 1) {
                        // pasable para buscar número por maplocation
                        // passable2 para buscar maplocation por el numero para el BFS
                                string = temp.getX() + "," + temp.getY();
                                passableTierra.put(string, contador);
                                passableTierra2.put(contador, string);
                                        if (planetaTierra.initialKarboniteAt(temp) > 1) {
        karboMapTierra.put(string, planetaTierra.initialKarboniteAt(temp));
                                        }
                                }
                                contador++;
                        }
                }


                adyTierra = relacionNodeTierra();
        }

        public static int[][] relacionNodeTierra() {

                MapLocation value;
                int X, Y;
                String string;
                int resultado[][] = new int[(nodoMaxTierra)][(nodoMaxTierra)];

                for (String key : passableTierra.keySet()) {
                        String[] parts = key.split(",");
                        X = Integer.parseInt(parts[0]);
                        Y = Integer.parseInt(parts[1]);
                        value = new MapLocation(Planet.Earth, X, Y);

                        for (Direction direction : directions) {

                                if (!direction.name().equalsIgnoreCase("Center")) {

string = value.subtract(direction).getX() + "," + value.subtract(direction).getY();

                                        if (passableTierra.containsKey(string)) {

                                        if (value.subtract(direction).getY() >= 0 &&
value.subtract(direction).getX() >= 0
                                                                        &&
value.subtract(direction).getY() < altoMapaTierra
                                                                        &&
value.subtract(direction).getX() < anchoMapaTierra) {


        resultado[passableTierra.get(key)][passableTierra.get(string)] = 1;
```

```
                                        }
                                    }
                                }

                            }
                    }

                    return resultado;

            }



            ////////////////////// bsf///////

            /**
             *
             * @param mapini
             * @param mapfin
             * @return null si ya el ini=fin o el fin ya está ocupado por otro.
             */
            public static Direction bfs(MapLocation mapini, MapLocation mapfin) {
                    String msg = "Ha llegado a su destino";
                    Planet planeta;
                    MapLocation value = null;
                    int prev[];
                    int V;
                    int ady[][];
                    List<Integer> path = new ArrayList<Integer>();
                    int ini, fin, pasos = 0, max = 0, actual;
                    boolean visitado[];
                    Direction directionToMove = Direction.Center;

                    HashMap<String, Integer> passable = new HashMap<String, Integer>();
                    HashMap<Integer, String> passable2 = new HashMap<Integer, String>();

                    if (mapini.getPlanet() == planetaTierra.getPlanet() && mapfin.getPlanet() ==
planetaTierra.getPlanet()) {

                            prev = new int[nodoMaxTierra];
                            V = nodoMaxTierra;
                            ady = adyTierra;
                            passable = passableTierra;
                            passable2 = passableTierra2;
                            visitado = new boolean[nodoMaxTierra];
                            planeta = planetaTierra.getPlanet();

                    } else {

                            prev = new int[nodoMaxMarte];
                            V = nodoMaxMarte;
                            ady = adyMarte;
                            passable = passableMarte;
                            passable2 = passableMarte2;
                            visitado = new boolean[nodoMaxMarte];
                            planeta = planetaMarte.getPlanet();
                    }


                    try {

                            String key1 = mapini.getX() + "," + mapini.getY();
                            String key2 = mapfin.getX() + "," + mapfin.getY();

                            Arrays.fill(visitado, false);

                            ini = passable.get(key1);

                            fin = passable.get(key2);
                            prev[ini] = -1;
                            visitado[ini] = true;
                            Queue<Integer> Q = new LinkedList<Integer>();
                            Q.add(ini);
                            while (!Q.isEmpty()) {
```

```
                                max = Math.max(max, Q.size()); // ver memoria usada en cola
                                actual = Q.remove();
                                pasos++; // número de vértices que estoy visitando
                                if (actual == fin)
                                        break;
                                for (int i = 0; i < V; ++i) { // vemos adyacentes a nodo
actual
                                        int v = ady[actual][i];

        if (v != 0 && !visitado[i]) { // no visitado agregamos a cola
                prev[i] = actual; // para ver recorrido más corto de nodo inicio a fin
                                                visitado[i] = true;
                                                Q.add(i);
                                                // }
                                        }
                                }
                        }

                        for (;path.size()<nodoMaxTierra;) {
                                path.add(fin);
                                if (prev[fin] == -1)
                                        break;
                                fin = prev[fin];
                        }

                        for (int i = path.size() - 1, k = 0; i >= 0; --i) {

                                if (k == 1) {

                                String[] parts = passable2.get(path.get(i)).split(",");
                                        int X = Integer.parseInt(parts[0]);
                                        int Y = Integer.parseInt(parts[1]);
                                        value = new MapLocation(planeta, X, Y);
                                        directionToMove = mapini.directionTo(value);
                                        msg = "Dirección de la unidad = " + directionToMove;
                                        break;
                                }
                                k = 1;

                        }

                } catch (Exception e) {
                        directionToMove = Direction.Center;
                        msg = "El terreno no es pasable";
                        System.out.println("Player.BFS = " + msg);
                }
                // System.out.println("Player.BSF = " + msg);

                /*
                 * Este if es para evitar que si la unidad no puede moverse hacia la dirección
                 * "directionToMove" debido a que está ocupado por una estructura o aliado.
                 */
                if (value != null && !checkPassable(value)) {
                        for (Direction direction : directions) {
                                if (!direction.name().equalsIgnoreCase("Center") &&
checkPassable(mapini.add(direction))) {
                                        directionToMove = direction;
                                        break;
                                }

                        }
                }
                return directionToMove;
        }

        public static MapLocation distance(MapLocation first, HashMap<String, Long> map) {
                int x1 = first.getX();
                int y1 = first.getY();
                int x2;
                int y2;
                String[] positionXY;
                int resultado;
                String resultadoKey = "";
                int resultadoAux = 0;
```

```
                int count = 0;
                MapLocation localizacionKarbonite;

                try {

                        for (String key : map.keySet()) {
                                count++;
                                positionXY = key.split(",");
                                x2 = Integer.parseInt(positionXY[0]);
                                y2 = Integer.parseInt(positionXY[1]);
                                if (count == 1) {
                                resultado = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
                                        resultadoKey = key;
                                        resultadoAux = resultado;
                                } else {
                                resultado = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
                                }

                                if (resultadoAux > resultado) {
                                        resultadoAux = resultado;
                                        resultadoKey = key;
                                }
                        }

                        positionXY = resultadoKey.split(",");

                        localizacionKarbonite = new MapLocation(Planet.Earth,
Integer.parseInt(positionXY[0]),Integer.parseInt(positionXY[1]));

                        return localizacionKarbonite;

                } catch (Exception e) {
                        return null;
                }
        }

        static public void contarUnidades(VecUnit units) {

                Unit unit;
                // Resetear las variables
                countFactory = 0;
                countHealer = 0;
                countKnight = 0;
                countMage = 0;
                countRanger = 0;
                countRocket = 0;
                countWorker = 0;

                for (int i = 0; i < units.size(); i++) {
                         unit = units.get(i);

                        if (unit.health() > 0) {
                                switch (unit.unitType()) {
                                case Factory:
                                        countFactory++;
                                        break;
                                case Healer:
                                        countHealer++;
                                        break;
                                case Knight:
                                        countKnight++;
                                        break;
                                case Mage:
                                        countMage++;
                                        break;
                                case Ranger:
                                        countRanger++;
                                        break;
                                case Rocket:
                                        countRocket++;
                                        break;
                                case Worker:
                                        countWorker++;
                                        break;
```

```
                                }
                        }
                }
        }

        // Localiza el enemigo
        public static MapLocation destino(Unit unit) {

                VecUnit unidadesEnemigasAux =
gc.senseNearbyUnitsByTeam(unit.location().mapLocation(), unit.visionRange(),
Utilidades.equipoEnemigo);

                if (unidadesEnemigasAux.size() != 0) {
                        resultado = unidadesEnemigasAux.get(0).location().mapLocation();
                }
                if (resultado == null) {
                        resultado = unit.location().mapLocation()
                        .add(directions[(int) Math.floor(Math.random() * (7 - 0 + 1) + 0)]);
                }
                return resultado;
        }

        // Mueve y localiza el enemigo si el argumento es direction=null
        public static void mover(Direction direction, Unit unit) {

        try {
                if (direction == null) {
                direction = Utilidades.bfs(unit.location().mapLocation(), destino(unit));
                        }

                        // mover el robot hacia la dirección.
if (gc.isMoveReady(unit.id()) && direction != null && gc.canMove(unit.id(), direction)) {

                                gc.moveRobot(unit.id(), direction);

                        }

                } catch (Exception e) {
                        return;
                }
        }

public static void atacar(Unit unit) {

        try {

        VecUnit enemigosCercanos = gc.senseNearbyUnitsByTeam(unit.location().mapLocation(),
unit.visionRange(),Utilidades.equipoEnemigo);
                        long size = enemigosCercanos.size();
                        for (int i = 0; i < size; i++) {

                                if (gc.canAttack(unit.id(), enemigosCercanos.get(i).id())) {
                                        gc.attack(unit.id(), enemigosCercanos.get(i).id());

        resultado=enemigosCercanos.get(i).location().mapLocation();
                                        break;
                                }

                        }
                } catch (Exception e) {
                        return;
                }

        }

        public static void producirRobot(Unit unit, UnitType unitType) {

                try {
                        if (gc.canProduceRobot(unit.id(), unitType)) {

                                gc.produceRobot(unit.id(), unitType);
                        }
```

```
                } catch (Exception e) {
                        return;
                }
        }

}
```

C. Clase Factoría

Código.

```
import bc.*;

public class Factory {

        static Unit unit;
        static GameController gc;

        public static void ejecutar(GameController gc, Unit unit) {

                Factory.unit = unit;

                Direction[] directions = Utilidades.directions;

                try {
                        VecUnitID garrison = unit.structureGarrison();

                        long size = garrison.size();

                        for (int i = 0; i < size; i++) {

                                // unload units
                                for (Direction direction : directions) {
                                        if (gc.canUnload(unit.id(), direction)) {
                                                gc.unload(unit.id(), direction);
                                                break;
                                        }
                                }
                        }

                        Utilidades.producirRobot(unit, UnitType.Ranger);

                } catch (Exception e) {
                        System.out.println("FACTORY: No se ha producido la unidad");
                }

                return;

        }

}
```

D. Clase Worker

Código:

```
import bc.*;
import java.util.*;

public class Worker {

	static Unit workerBuid;

	static int limiteFactory = 3;

	static int idFactoryBluePrint;

	public static void ejecutar(GameController gc, Unit unit) {

		Direction[] directions = Utilidades.directions;
		MapLocation workerLocation = unit.location().mapLocation();

		MapLocation karboniteLocation = Utilidades.distance(workerLocation,
Utilidades.karboMapTierra);

		Direction direction = Utilidades.bfs(workerLocation, karboniteLocation);

		//Visualizar enemigo
		Utilidades.destino(unit);

		if (gc.round() == 1) {
			workerBuid = unit;
			for (int i = 0; i < directions.length; i++) {
				if (gc.canReplicate(unit.id(), directions[i])) {
					gc.replicate(unit.id(), directions[i]);
					break;
				}
			}
		}

		if (workerBuid.health() <= 0) {
			workerBuid = unit;
		}



		if (unit.id() == workerBuid.id()) {

			if (gc.canBuild(unit.id(), idFactoryBluePrint)) {
				gc.build(unit.id(), idFactoryBluePrint);
				return;
			}

			for (int d = 0; d < directions.length; d++) {
			if (gc.canBlueprint(unit.id(), UnitType.Factory, directions[d])
					&& Utilidades.countFactory < limiteFactory) {
				gc.blueprint(unit.id(), UnitType.Factory, directions[d]);

idFactoryBluePrint =
gc.senseUnitAtLocation(unit.location().mapLocation().add(directions[d])).id();
				}

			}

			return;
		}
```

```
                if (gc.canHarvest(unit.id(), direction)) {
                        gc.harvest(unit.id(), direction);

                        try {
                                if (gc.karboniteAt(karboniteLocation) == 0) {

        Utilidades.karboMapTierra.remove(karboniteLocation.getX() + "," +
karboniteLocation.getY());
                        direction = Utilidades.bfs(workerLocation, karboniteLocation);
                                }
                        } catch (Exception e) {
                                Utilidades.karboMapTierra.remove(karboniteLocation.getX() +
"," + karboniteLocation.getY());
                        }

                }

                Utilidades.mover(direction, unit);

        }

}
```

E. Clase Ranger

Código

```
import bc.*;

import java.util.*;

public class Ranger {

        public static void ejecutar(GameController gc, Unit unit) {

                Direction direction = null;

                if (Utilidades.countRanger < 10) {

        direction = Utilidades.directions[(int) Math.floor(Math.random() * (7 - 0 + 1) + 0)];

                        Utilidades.atacar(unit);

                } else {

                        try {
                                Utilidades.atacar(unit);
                                //Visualizar enemigo
                                Utilidades.destino(unit);
                        } catch (Exception e) {
                                // TODO: handle exception
                        }


                }



                //Mueve y localiza el enemigo
                Utilidades.mover(direction, unit);

        }

}
```