Universidad Carlos III de Madrid



Escuela Politécnica de Leganés

Inteligencia Artificial en la Industria del Entretenimiento

Tutorial 1

Grado en Ingeniería Informática

Curso 2017/2018

Autores

|  |  |  |
| --- | --- | --- |
| *Sara Yuste Fernández Alonso* |  | **100330038** |
| *Stefan Borgstein* |  | **100379586** |

Contenido

[1 Introducción 3](#_Toc494281471)

[1.1 Alcance 3](#_Toc494281472)

[2 Cuestiones 4](#_Toc494281473)

[3 Código implementado 6](#_Toc494281474)

[3.1 Estructuras de datos 6](#_Toc494281475)

[3.2 Métodos 6](#_Toc494281476)

[4 Conclusiones 8](#_Toc494281477)

# Introducción

El presente documento contiene la memoria del desarrollo de un agente automático que realice una serie de funciones en la plataforma *Starcraft*.

## Alcance

En primer lugar, se presenta una breve introducción a los contenidos a tratar a lo largo del documento.

Seguidamente, se procede a responder las cuestiones 4,5,6,7,8,9 y 15 del enunciado del tutorial

En tercer lugar, se describen las estructuras de datos y métodos empleados en la resolución de los apartados 11, 12 y 13.

Por último, se exponen una serie de conclusiones y opiniones personales del grupo de prácticas tras la realización del tutorial.

# Cuestiones

**4.**

***OnStart***: Este método obtiene los valores necesarios para comenzar el juego, lee el mapa y analiza su información.

***OnFrame***: Contiene el bucle principal del juego. Obtiene la información del jugador, almacena las unidades en una estructura de datos, y dependiendo del tipo de unidad realiza las operaciones pertinentes. También se encarga de entrenar a los constructores y ponerles a trabajar.

***OnEnd***: Este método es llamado al finalizar la partida, y se encarga de notificar la finalización de la misma al usuario.

**5.**

onUnitCreate maneja la creación de unidades, luego este evento se produce al querer crear una unidad. En principio, en el código visible por los alumnos, no realiza ninguna función.

**6**.

onUnitComplete se ejecuta al terminarse la creación de una unidad, luego este evento se produce al finaizar la creación de una unidad. En el código visible por los alumnos, este método notifica de la creación de la unidad.

**7.**

Los métodos relativos BWTA (BWTA.getRegions(), por ejemplo), manejan información del mapa en conceptos de regiones. Esto es importante ya que internamente el mapa funciona de ese modo, pero para el usuario es más cómodo trabajar con casillas (tiles) o píxeles. Para eso, se hacen uso de los métodos de Game (game.mapHeight(), game.mapWidt()) y la constante TILE\_SIZE para convertir píxeles en casillas y viceversa.

**8.**

IsBuildable: dada una posición del mapa, devuelve la altura del terreno en esa posición. Esto es importante ya que no se puede construir en todas las alturas.

canBuildHere: dadas una unidad y una casilla del mapa, devuelve un valor booleano dependiendo de si la unidad dada puede construir en la casilla dada.

**9.**

isWalkable: dado que un agente no puede caminar libremente por todas las casillas del juego, este método, dada una casilla, devuelve un valor booleano dependiendo de si se puede caminar sobre dicha casilla.

**15.**

* No se ha encontrado ningún método que permita obtener este dato, ya que los métodos de posición devuelven la posición superior izquierda (bien sea el píxel o la casilla superior izquierda).
* Sí, utilizando los métodos isConstructed(), isBeingConstructed().
* El tamaño máximo de un edificio Terran es 117x83 (tamaño real). Este tamaño corresponde al centro de comando, el edificio más grande de la raza Terran.
* Sí, utlizando los métodos isConstructing(), isCarryingGas(), isCarryingMinerals(), isGatheringGas(), isGatheringMinerals(), isIdle().
* Para saber qué unidades pertenecen al jugador ejecutado por el bot, es necesario utilizar la clase self(). Así, si se recorren las unidades de esa clase (for Unit myUnit: self.getUnits()), y se compara el ID de dicha unidad con los IDs de las unidades propias, se podrá saber si la unidad pertinente pertenece al jugador del bot.
* Para ello deberá iterarse sobre las unidades del jugador (como se explica en el apartado anterior), y para cada una utilizar los métodos: getType(), getTilePosition() (o getPosition()), isGatheringGas(), isGatheringMinerals(), isIdle(), isCompleted(), isConstructed(), isBeingConstructed(), isConstructing(), isCarryingGas(), isCarryingMinerals()

# Código implementado

## Estructuras de datos

Para el manejo de las unidades de trabajadores, se crearon tres listas:

* Trabajadores ociosos: public List<Unit> IdleWorkers = new ArrayList<Unit();
* Trabajadores recogiendo minerales o gas: public List<Unit> BusyWorkersMineralsOrGas = new ArrayList<Unit();
* Trabajadores construyendo un depósito de suministros: public List<Unit> BusyWorkersSupply = new ArrayList<Unit();

El objetivo de estas listas es conocer en cada momento en qué estado se encuentran los trabajadores del usuario. En principio se crearon dos listas separadas para trabajadores recolectando gas y minerales, pero finalmente se decidió que la lógica era similar y no era necesario crear dos listas para este propósito.

Cada vez que se completa una unidad, en el método onUnitComplete(), se comprueba si la unidad es propia, y si es un trabajador. En ese caso, se añade la unidad a la lista de trabajadores ociosos. En el método principal de juego onFrame(), se retiran los trabajadores ociosos de esta lista y se colocan en las otras listas según la tarea que les sea asignada.

## Métodos

Los métodos creados por los alumnos son:

**buildSupplyDepot**: Dada una unidad, primeramente la añade a la lista de unidades que se encuentran construyendo un depósito de suministros. Seguidamente, se obtiene la posición desde donde se comenzará a construir el edificio. Para esto, se hace uso del método findLocationToBuild, implementado por los alumnos y que será descrito en apartados sucesivos.

A continuación, se comprueba si la unidad seleccionada para construir puede efectivamente construir ese edificio en esa posición, utilizando el método canBuild(). Se ha hecho uso de este método ya existente debido a que facilita el proceso de comprobación que de no utilizarse debería implementarse por los alumnos. En caso de que pueda construir el edificio, se llama la función buidingTrain, implementada por los alumnos. En caso contrario, se muestra un mensaje de error.

**buildBarrack**: Este método no recibe la unidad que va a construir, sino que utiliza como constructor a algún trabajador que esté recolectando mineral o gas. Para obtener la posición donde debe construir, hace uso de la función findLocationToBuild(). Comprueba si la unidad puede construir el edificio en esa posición y en caso afirmativo, llama a buildingTrain, imprimiendo un mensaje explicativo. En caso contrario, imprime un mensaje informativo.

**buildRefinery**: Este método recibe la unidad que va construir la refinería. Itera sobre las unidades neutrales del mapa, y cuando encuentra una que no sea un campo de minerales (y por tanto, es un geyser de gas), obtiene su posición mediante el método getTilePosition(), y comprueba si puede construir la refinería en esa posición. En caso afirmativo, manda a la unidad a recolectar gas y la añade a la lista de trabajadores ocupados recolectando.

**gatherMinerals**: Funciona de forma similar al método anterior, con la diferencia de que no es necesario construir un edificio específico para recolectar minerales. Por ello, una vez encontrada la mina más cercana, manda a un trabajador a recolectar minerales, y le añade a la lista de trabajadores ocupados recolectando.

**findLocationToBuild**: Este método obtiene la localización donde construir un edificio basada en la posición relativa del centro de comando a las minas. Así, compara las coordenadas x e y (haciendo uso del método ya implementado getX(), getY()) del centro de comando con las coordenadas de la fuente de minerales más cercana. El objetivo es construir el edificio en dirección contraria a la posición de la mina con respecto al centro de comando, para evitar que los trabajadores tengan que rodear un edificio para recolectar mineral. Así, si la mina está al este del centro de comando, se fijarán las coordenadas base desde donde construir al oeste del centro de comando; y así sucesivamente con los 8 principales puntos cardinales (norte, sur, este, oeste, noreste, noroeste, sureste, suroeste). El método devuelve la casilla donde construir, formada por las coordenadas base obtenidas.

**unitTrain**: Recibe el tipo de unidad y el edificio donde será entrenada. Se utiliza para construir dos tipos de unidades; marines y trabajadores. Para ello, comprueba que el tipo de unidad coincide con el tipo de edificio donde debe ser entrenada, y se hace uso de la función train() ya implementada.

**buildingTrain**: recibe la unidad que va a construir el edificio, el edificio que se desea construir, y la posición donde se desea construir. Con estos tres parámetros, haciendo uso del método build() de la clase Unit ya implementado, se ordena a la unidad proporcionada que construya el edificio.

Todos estos métodos son llamados en el método onFrame(). La primera premisa de este método y su prioridad es mandar trabajadores a recolectar minerales y gas. Para ello, los irá obteniendo de la lista de trabajadores ociosos, y asignando tareas. En caso de que no se haya alcanzado el número máximo de trabajadores (10), se entrenarán más unidades del tipo Terran\_SCV. Una vez se haya alcanzado ese número, y si se disponen de 150 minerales, se construirá un cuartel. Si se ha construido un cuartel en algún momento, se entrenarán marines en él. Por último, si no se disponen de suministros suficientes, y se tienen 100 minerales, se construirá un depósito de suministros. Esta tarea se le asignará a un trabajador ocioso, si lo hay, y si no a alguno de los recolectores de gas o minerales.

# Conclusiones

El equipo de prácticas ha encontrado muchas dificultades en el desarrollo del presente tutorial. La primera de ellas, y la que más tiempo ha consumido, ha sido la correcta instalación de BWAPI y los componentes necesarios para el desarrollo de la práctica. Esta plataforma daba errores que en algunos casos ha sido imposible identificar adecuadamente y solucionar. En uno de los ordenadores utilizados por la pareja de prácticas, no conseguía asociarse correctamente a Starcraft, dado que no encontraba el directorio donde estaba situado (a pesar de que éste hubiera sido seleccionado manualmente), con lo que no se creaba el directorio bwapi-data dentro de /Starcraft. Debido a esto, ChaosLauncher era incapaz de ejecutarse, ya que carecía de la información necesaria para hacerlo. Este problema no pudo solucionarse y se tuvo que hacer uso de otro ordenador.

Además, el uso de java de 32 bits ha supuesto un problema menor: en la primera instalación llevada a cabo, Eclipse pudo abrir y editar el proyecto sin problemas. Sin embargo, al apagar el ordenador e intentar volver a abrir el proyecto en Eclipse, el programa no era capaz de iniciar. Este problema se resolvió desinstalando la versión de Eclipse de 64 bits e instalando la de 32 bits.

En cuanto al desarrollo del código, la estructura de control de Starcraft ha hecho que tuviera que cambiarse por completo la lógica del programa, lo cual no era evidente en un primer momento. A pesar de que la lógica original parecía correcta, Starcraft tiene una lista de prioridades internas, según la cual coloca a las unidades trabajando dándole prioridad a recolectar mineral, y, lo que es más importante, sin la creación de las listas descritas era imposible mandar a una unidad que se encontraba recolectando minerales a recolectar gas. Esto es debido a que el propio motor de Starcraft manejaba esa unidad internamente de forma invisible para el programador. Este problema se solucionó con la creación de las listas.

Otra dificultad encontrada fue la longitud de la documentación referente a BWAPI y BWMirror, que supuso que los alumnos tuvieran que invertir mucho tiempo en conocerla. Aun así, una vez invertido ese tiempo, esta dificultad se convirtió en ayuda, gracias a los múltiples métodos ya implementados con funcionalidades muy útiles para el tutorial..

En resumen, el grupo de prácticas opina que este tutorial, si bien sirve para familiarizarse con la plataforma, es difícil de llevar a cabo en el tiempo dado.