**Inteligencia Artificial para Videojuegos**

***Práctica 4: Frente de guerra zombie***



*Grupo 07*

*Alejandro Ortega Álvarez*

*Borja Cano Álvarez*

Problema a resolver

* La propuesta es representar en Unity un escenario o mapa (mediante casillas) donde se sitúa un agente y (opcionalmente) aliados que ayudan al agente para combatir una oleada de zombies.



* Para ello se piden los siguientes requisitos:

1. Un tablero de 6x12 (posiblemente modificable).
2. Al empezar, se sitúan el agente y el refugio (en este caso es una nave), en las casillas 0,0 y 0,1.
3. Después el usuario puede poner cualquier cantidad de aliados y zombis distribuidos por el resto del mapa (hasta el límite que haya puesto, en este caso se pueden máx. 5 aliados y 20 zombies)
4. Una vez habiendo colocado las entidades como se desee, dándole al botón comenzar, empezará el simulador de batalla por turnos (los aliados no participan pero ayudan a que el agente se haga con la victoria).
5. El modo noche se puede activar y desactivar en cualquier momento de la partida e influye en el desarrollo de la misma.
6. El juego acaba si el agente muere, si vuelve a la nave o si consigue acabar con todos los zombies.

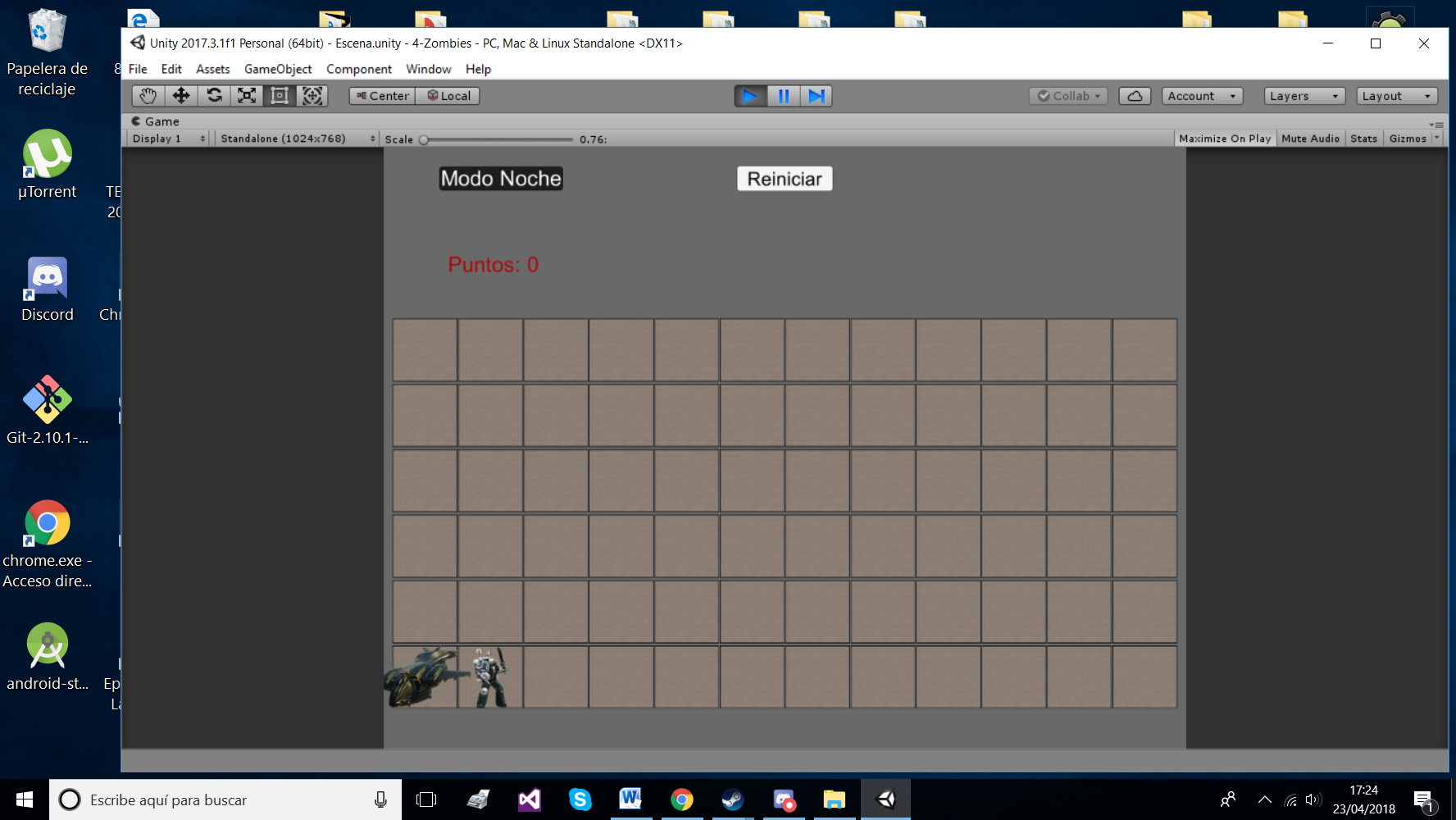
**Adicionales:**

1. Botón para reiniciar simulador en cualquier momento.
2. Sólo se puede comenzar la partida si hay al menos un zombie en el mapa.
3. Posibilidad de hacer jugar al usuario. Las decisiones del agente (Atacar, Retroceder o Esperar) las puede decidir tanto el ordenador mediante el algoritmo o el usuario según le convenga.
4. Texto que muestra los puntos y el turno actual que corresponda.

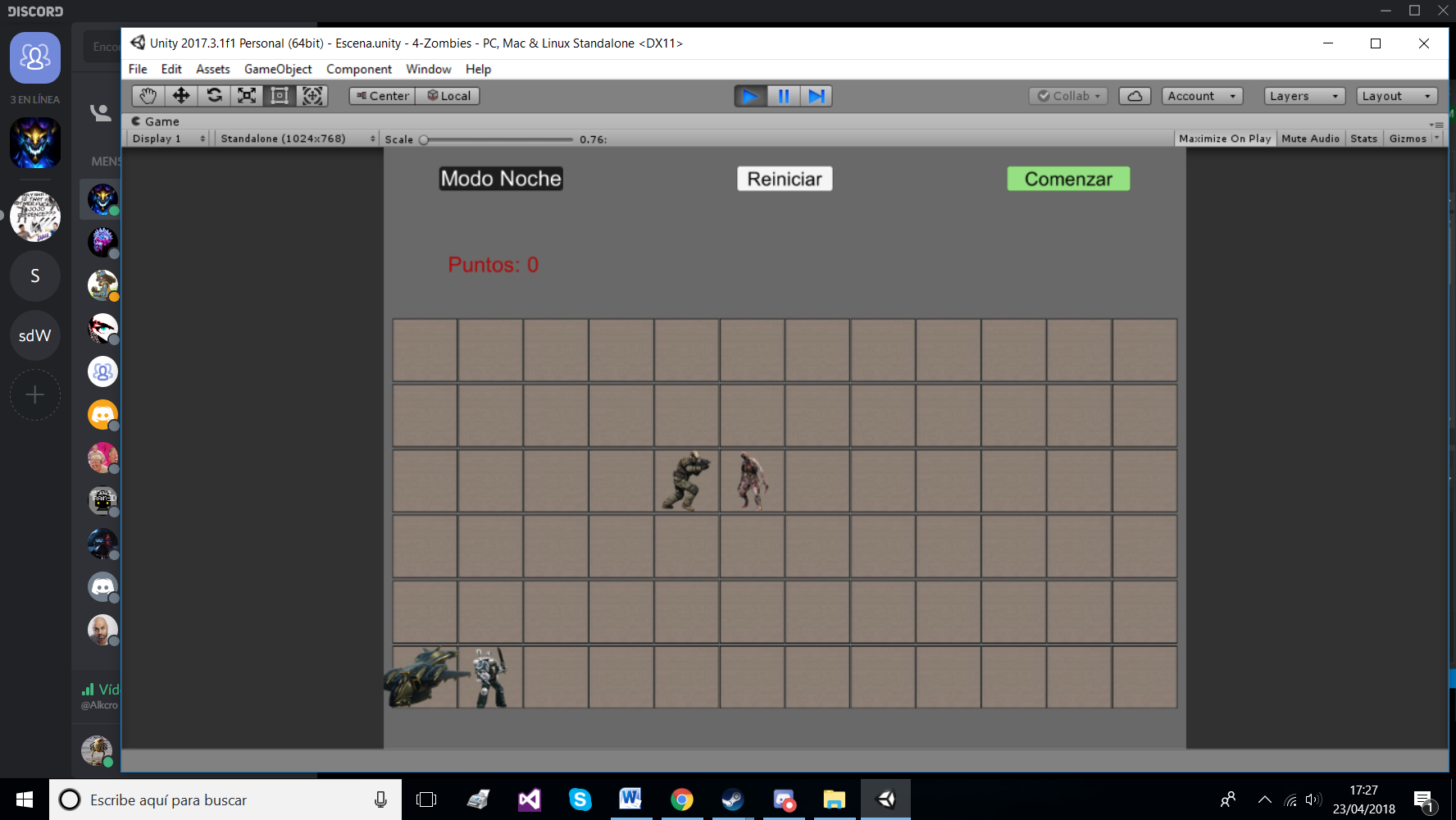
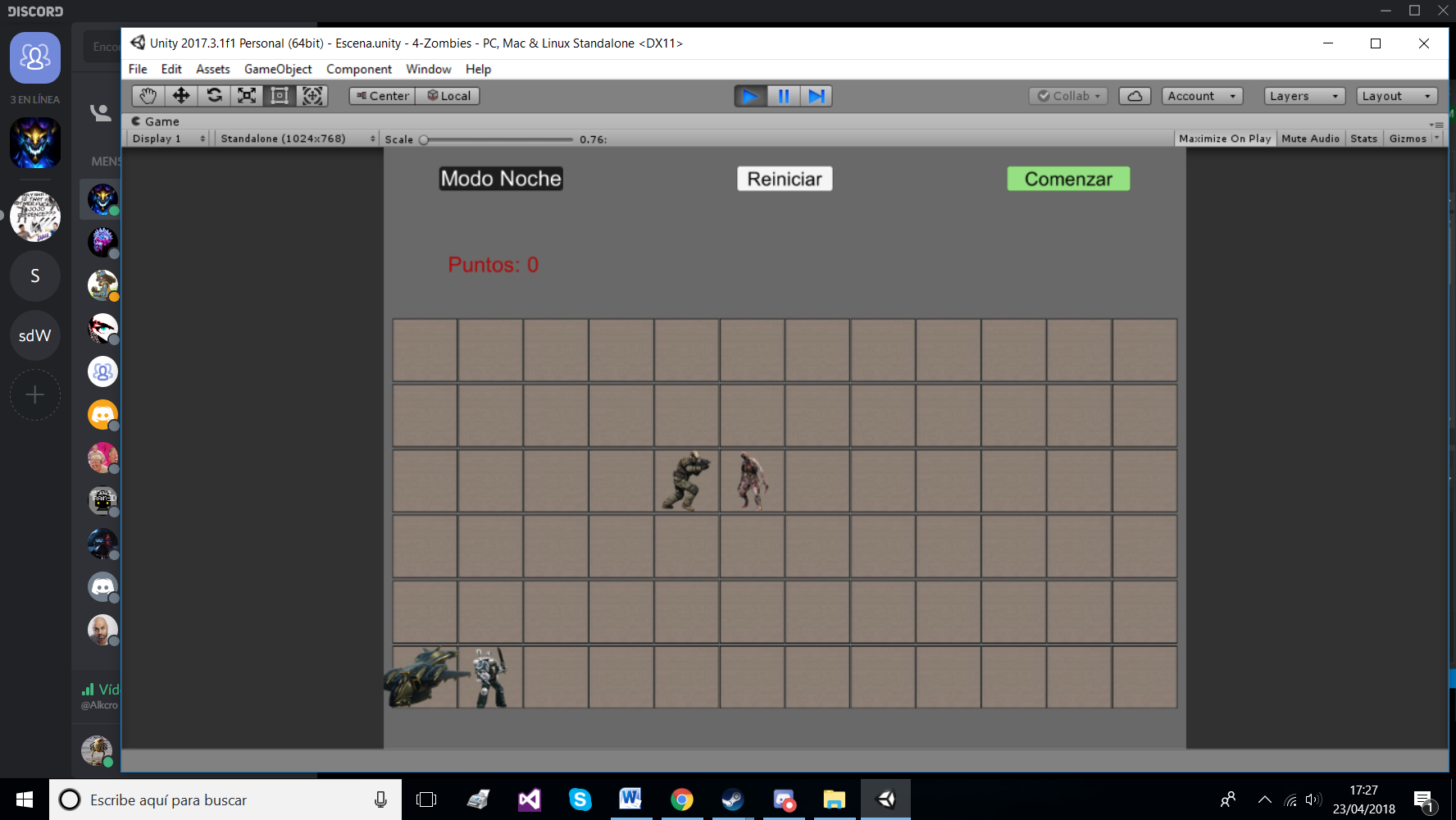
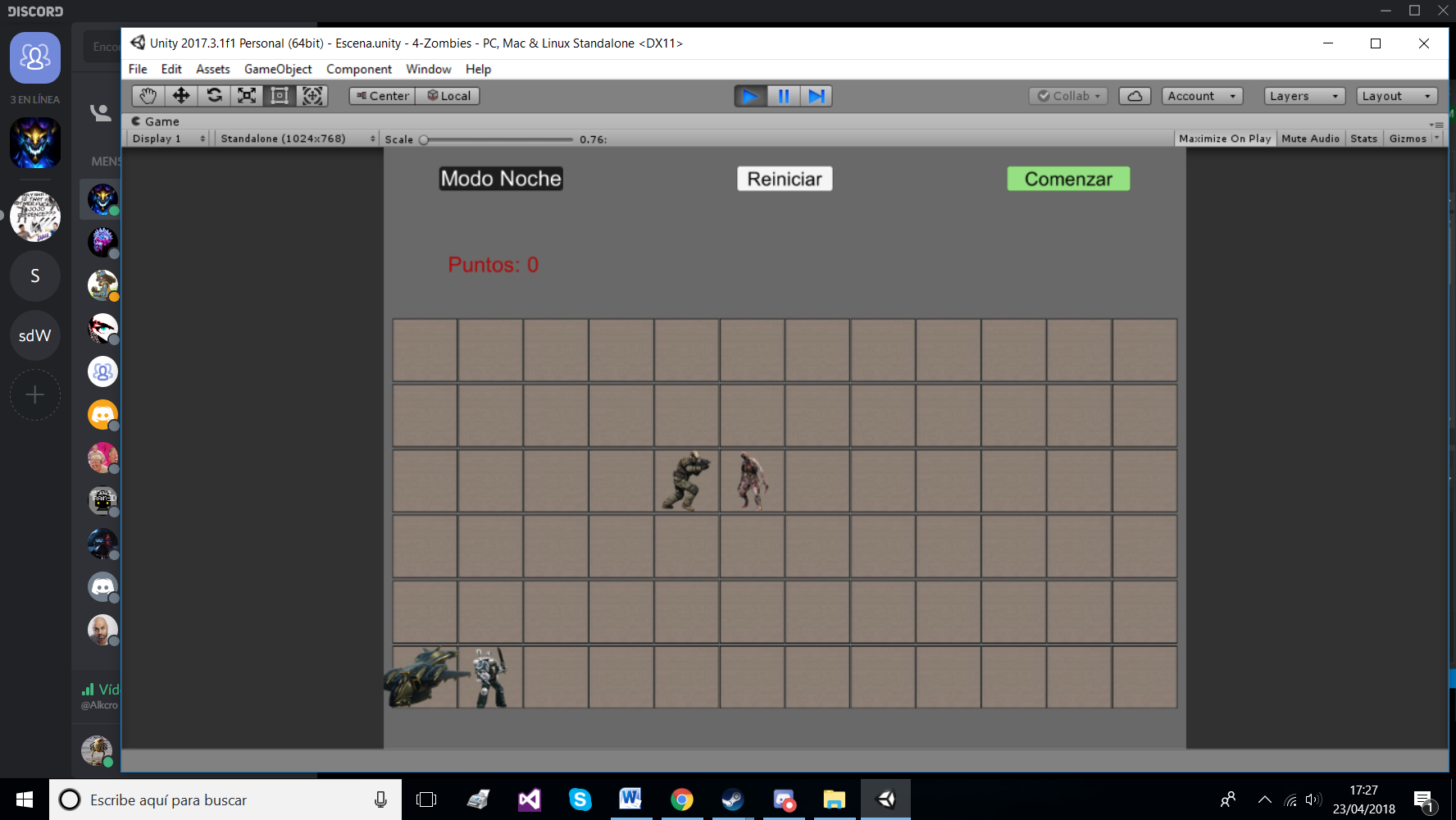
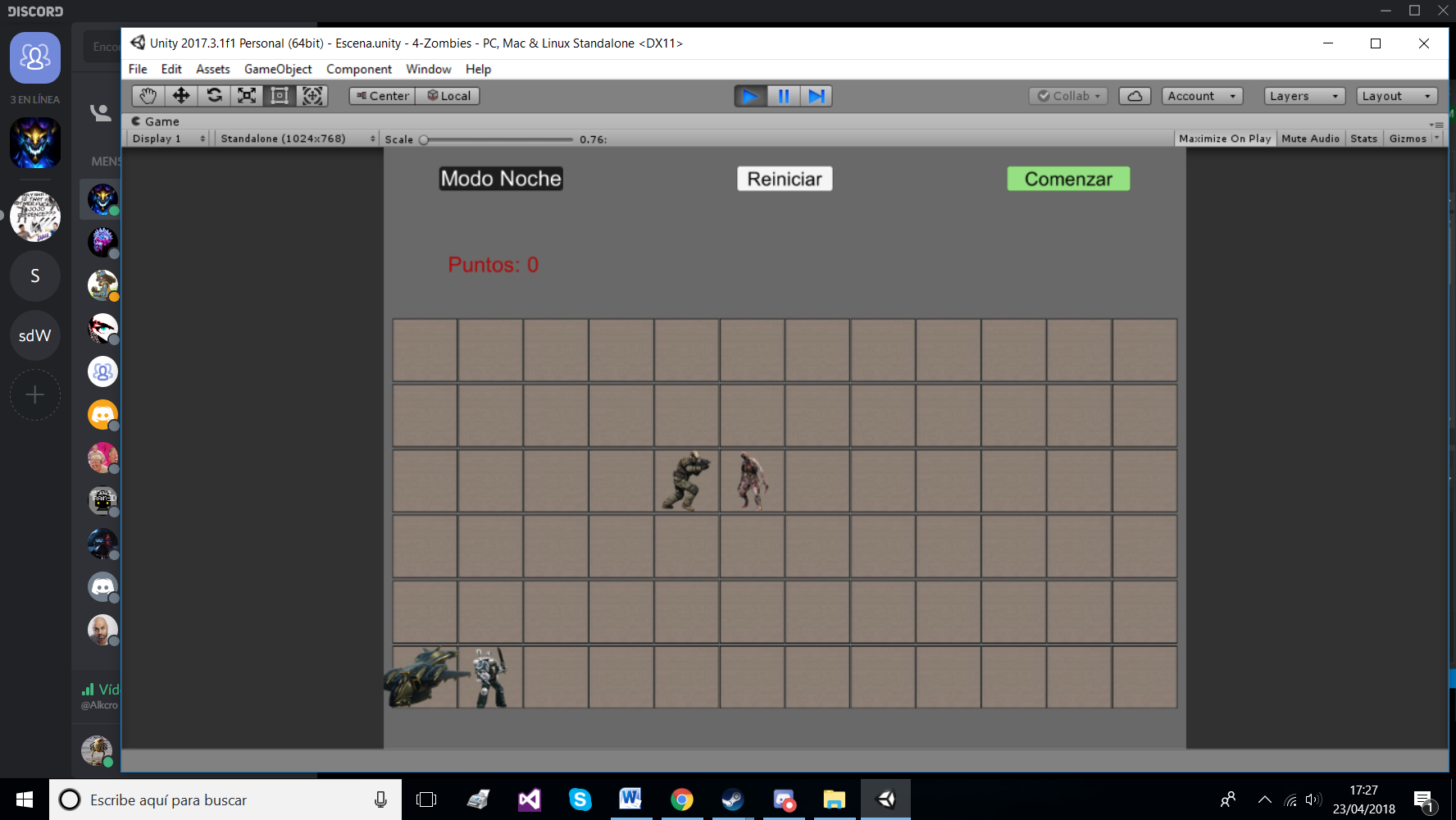
El juego

Repasamos el funcionamiento final del simulador.

Ésta es la vista nada más iniciar el juego:



Al presionar cualquier casilla (excepto las que están ocupadas por la nave y el agente) aparece un aliado y si se vuelve a hacer click en su lugar aparece un zombie.

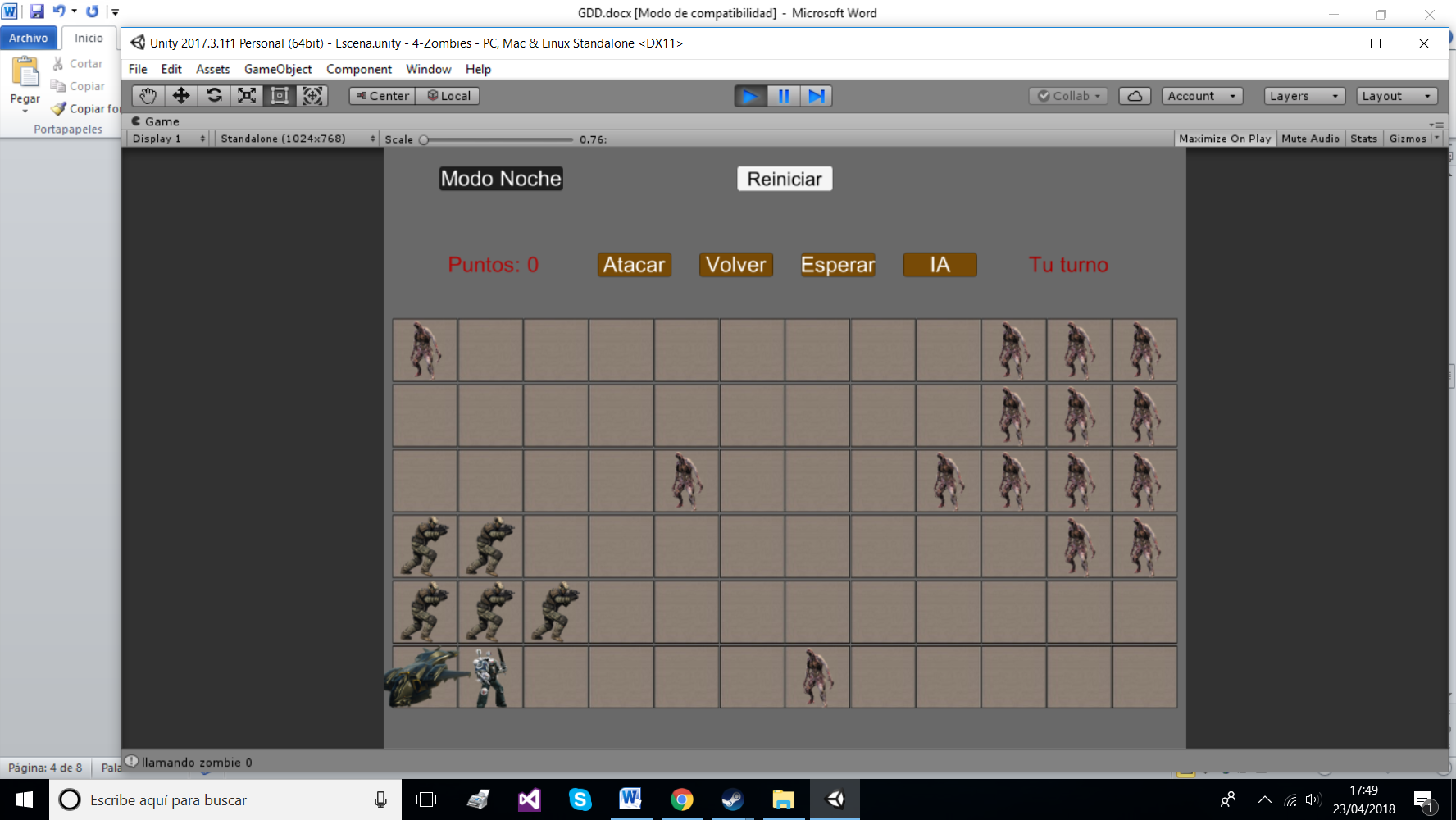


Una vez haya al menos un zombie en el mapa, se puede comenzar la batalla. La batalla se desarrolla por turnos, el turno de los zombies y el turno del agente. El primer turno se decide de forma aleatoria.

En cualquier momento de la partida se puede reiniciar o cambiar entre modo noche/dia, pero no se puede clickear una casilla para añadir entidades si se ha comenzado ya la batalla.

**El turno del agente**

En el turno del agente se muestra el siguiente menú de acciones:



El usuario decide qué acción hace el agente o si lo decide la IA por él.

Tal como en el enunciado de la práctica, las acciones para cada turno son:

1. **Atacar** al zombie más cercano (avanza una casilla hacia él y hay batalla si entra en la misma casilla).
2. **Retroceder** una casilla hacia la nave.
3. **Esperar** y permanecer en el sitio.

**El turno de los enemigos**

Los zombies sólo buscan atrapar a los agentes y acabar con ellos. La única acción que realizan por lo tanto es avanzar una casilla hacia el agente más cercano o aliado, y atacar si entra en la misma casilla. Cada zombie tiene su turno independiente y espera a que haya terminado el turno del zombie anterior.

Los aliados **no** entran en los turnos, se quedan en la posición que estén para el resto de la partida.

**Los combates**

Los combates sólo tienen lugar cuando un zombie entra en la casilla de algún soldado o el héroe en la casilla de uno (o varios) zombies. En los combates se decide, mediante probabilidades, quién vive y quién muere en el conflicto.

Las probabilidades dependen de dos factores:

* La **situación**: cantidad de enemigos vs. cantidad de aliados. Es más probable que gane el héroe si dispone de más apoyo por parte de los aliados.
* La **destreza**: cantidad de aliados vs. visibilidad (si es o no modo noche).

Dependiendo de estos factores, puede suceder que muera el atacante, el atacado o ambos. Si hay varios atacados en una misma casilla se hacen los combates por separado.

**Puntuación**

Cada resultado de un combate se ve reflejado en la puntuación de la siguiente forma:

|  |  |
| --- | --- |
| **Si un aliado mata un zombie** | 1 punto |
| **Si el agente mata un zombie** | 5 puntos |
| **Si muere un aliado** | -10 puntos |
| **Si muere el héroe** | -50 puntos |

La IA del agente debe decidir la acción dependiendo tanto de los puntos como de la situación.

\*La puntuación máxima que se puede conseguir con 20 zombies es de 100 puntos. (Victoria arrasante)

\*La puntuación mínima con 5 aliados es de -100 puntos. (Fracaso total)

**Cuándo acaba la partida**

La partida acaba si muere el agente, mueren todos los zombies o se retira el agente (consigue llegar a la nave antes de que le acorralen los zombies)

Implementación

* El Script de IdCasilla se encarga de devolver el identificador que posee cada casilla, con un enumerado que hace referencia a cada tipo.
* El Script de GameManager gestiona todos los cambios de estado del juego (creación inicial del mundo, realizar un asesinato, colocar agujeros y reiniciar).
* El Script de Patrulla se encarga de gestionar los distintos comportamientos del agente antes de haber resuelto el crimen, explicados anteriormente (patrón de patrulla inicial, encontrar sangre, encontrar barro, encontrar arma y cadáver).
* El script de Algoritmo devuelve una lista con el camino más corto a casa desde la posición del detective entre las casillas que actualmente estén descubiertas
* El Script de VueltaAcasa se llama al resolver el crimen y llama al Algoritmo para que le devuelva el camino de vuelta a casa y lo procesa para darle las velocidades correspondientes.

Algoritmo

**Problemas**

-El problema más importante que nos ha ocurrido ha sido el mismo que para la práctica de los tanques: la posición exacta por el tablero del detective, dado que le aplicamos una velocidad y su posición en sí misma muchas veces no va a coincidir con la de un entero por una millonésima de unidad.

Esto nos obliga a redondear y/o truncar la posición, y pronto nos dimos cuenta que esto dependía de la dirección de la velocidad (si se va para abajo o a la derecha, hay que truncar la posición, a la inversa hay que redondearla). Con esto ya casi lo teníamos, pero nos seguían ocurriendo cosas raras y después de varios días investigando, descubrimos que sólo modificábamos la posición en X (con respecto a si es truncamiento o redondeo) si había cambios en la dirección en X, y lo mismo para la Y, por lo que dependía completamente de la suerte si en el momento de cambio de dirección en X, la Y era correcta siendo truncada o redondeada.

En resumen, había que guardar el estado de cambio de dirección anterior tanto para X como para Y, para saber si truncar o redondear cuando vale 0 la velocidad en ese eje.

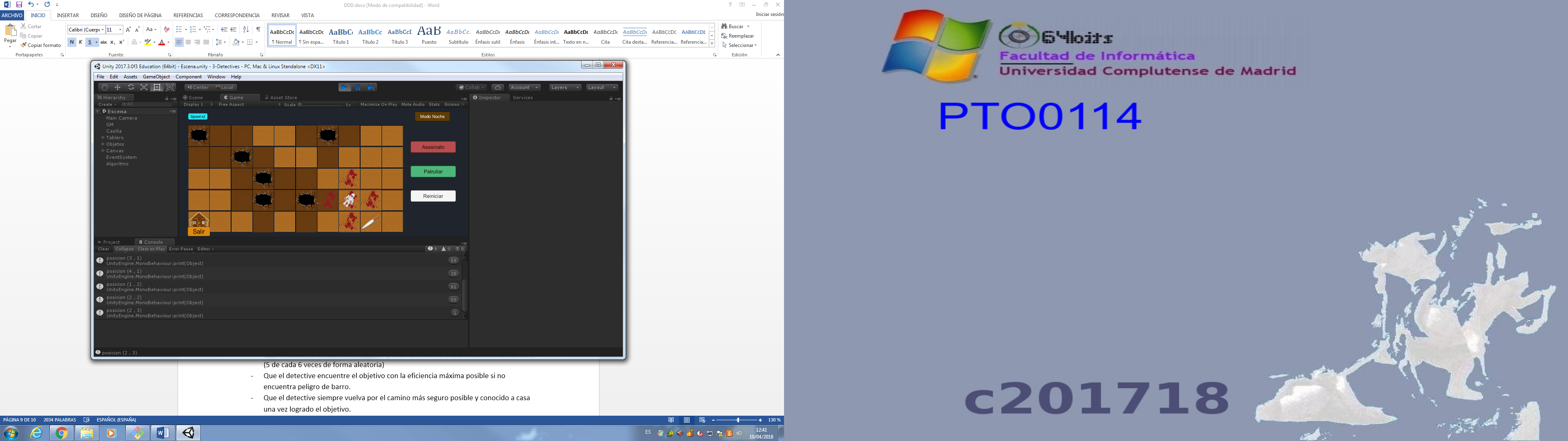
Aun así, aunque ya funcionan bien las posiciones y no se sale nunca de rango, en ocasiones esporádicas, al pasar al lado de un agujero sin tocarlo o del arma, hace como que pasa por encima, y creemos que sigue siendo debido a algo de esto.

-Hubo inconvenientes a nivel de código, al haber un montón de opciones posibles en la patrulla del agente (si encuentra sangre cambia el modo de búsqueda, lo mismo si encuentra el arma, si ya ha acabado, si encuentra barro, si ha muerto, etc.) y eran muchas líneas de código. Pero más o menos lo conseguimos abstraer todo con un enumerado que controlara el estado del agente para saber qué debía hacer, y con eso nos ahorramos unas cuantas líneas de código.

-Otro problema notable fue a la hora de decidir qué debía hacer el agente al encontrar barro. Discutimos varias opciones, que rodeara siempre en un gran área, que pasara y siguiera con la patrulla a ver si había suerte, que decidiera de forma aleatoria qué dirección tomar al llegar a una casilla de barro (era una buena opción pero muy costosa ya que requería reestructurar el sendero que habíamos puesto fijo y rompía los esquemas de la patrulla). Al final se decidió que si accedía a una casilla embarrada no se la podía jugar yendo a una casilla desconocida, por lo que rompía los esquemas de patrón de búsqueda inicial pero al menos ahora la mayoría de los mapas los puede pasar.

-El último problema (que de hecho ha sido el único que no hemos podido solucionar al 100%) es el localizar que el agente se queda encerrado en casillas embarradas, de este estilo:

De esta forma, el detective se volvía loco porque nunca accedía a una casilla desconocida cuando tocaba barro. Lo ideal sería detectar cuando se quedaba encerrado, pero nos resultó imposible. Lo único que se nos ocurrió es darle un valor random en el que 1 de cada 8 veces, el detective se la jugaba a acceder a una casilla desconocida estando el barro. Solucionábamos el problema parcialmente, ya que es muy probable que si se queda encerrado alguna de las veces que toca barro se la juega y puede continuar el desafío. Pero sabemos que no es la forma más eficiente de solucionarlo.



**Resultados**

Al final hemos logrado lo siguiente:

* Que el mapa se genere tal como se pide, pudiendo colocar agujeros.
* Que se pueda cambiar al modo noche para más verosimilitud y que se pueda incrementar la velocidad.
* Que el detective encuentre el cadáver y el arma entorno a una estadística del 85-90% (5 de cada 6 veces de forma aleatoria)
* Que el detective encuentre el objetivo con la eficiencia máxima posible si no encuentra peligro de barro.
* Que el detective siempre vuelva por el camino más seguro posible y conocido a casa una vez logrado el objetivo.

NO hemos logrado los siguientes objetivos (que reúnen el 10-15% de los casos en los que el detective no encuentra el cadáver):

* Que el detective consiga salir cuando está rodeado de barro de la forma más eficiente. Además, el recorrido por las casillas descubiertas una vez encontrado barro es completamente random salvo si encuentra casillas desconocidas, que les da preferencia. De esta forma, a veces debido al random da vueltas innecesarias. Probamos con una matriz que contara el número de veces que se pasaban por las casillas para que pasara por las que tuviera menor número, pero no nos llegó a funcionar.
* El detective en ocasiones esporádicas muere cuando pasa por una casilla cercana a un agujero. Esto se debe a un fallo que por más que depuramos no conseguimos localizar, aunque sabemos lo que ocurre no sabemos el por qué. Pocas veces pero las suficientes como para fastidiarnos el simulador, el método que devuelve la posición del detective se “escapa” un frame y da una posición que no es, sólo lo hace un frame y parece como si lo hiciera al azar. Si coincide justo ese frame puede dar una casilla que no es y morir en un hueco. De la misma forma puede ocurrir que el cuchillo o el arma lo coja sin estar en su casilla. Ocurre aproximadamente en torno a un 10% de las ocasiones.

Referencias y apoyos del desarrollo de la práctica:

* *Apuntes del Campus Virtual*
* *Google Imágenes*
* [*http://buildnewgames.com/astar/*](http://buildnewgames.com/astar/) *(algoritmo de la práctica anterior)*

*Repositorio de Github:* https://github.com/alex97ortega/IA.git