[75.07 / 95.02]

Algoritmos y programación III

Trabajo práctico 2: AlgoStar

(trabajo grupal)

Estudiantes:

Nombre	Padrón	Mail

Tutor:

Nota Final:

Índice

1. Objetivo

Desarrollar una aplicación de manera grupal aplicando todos los conceptos vistos en el curso, utilizando un lenguaje de tipado estático (Java) con un diseño del modelo orientado a objetos y trabajando con las técnicas de TDD e Integración Continua.

2. Consigna general

Desarrollar la aplicación completa, incluyendo el modelo de clases, sonidos e interfaz gráfica. La aplicación deberá ser acompañada por pruebas unitarias e integrales y documentación de diseño.

3. Especificación de la aplicación a desarrollar

AlgoStar es un juego similar al famoso StarCraft pero no en tiempo real sino por turnos. El mismo es un juego de guerra de estrategia y se basa en la construcción y administración de un imperio.

Las razas

- **Zerg**: Una raza de insectoides alienígenas en busca de la perfección genética, obsesionada con la asimilación de otras razas.

- **Protoss**: Son una especie humanoide con tecnología avanzada y habilidades psiónicas, tratando de preservar su civilización.

Gestión de recursos

Durante el juego se deberá recolectar:

- **Minerales**: habrá nodos de minerales distribuidos por todo el mapa, cada nodo tiene 2000 unidades de minerales.

- **Gas vespeno**: habrá volcanes que expulsan dicho gas cerca de los nodos de minerales. Cada volcán tiene 5000 unidades de gas.
  Para poder colectar el gas vespeno cada raza deberá construir una “refinería de gas” sobre el volcán dónde se origine el gas vespeno que desea extraer.

Construcciones

Zerg

Todas las unidades y “construcciones” zerg, son seres vivos biológicos, con lo cual, cuando las mismas son dañadas, se recuperan lentamente con el paso del tiempo hasta sanar al 100 %. Los zerg no tienen constructores, sino que sus larvas evolucionan a zángano y éste ser es el que muta en forma de “construcción”. Los zerg no crean unidades en sus “edificios”. A diferencia de los protoss, todas las unidades zerg se crean a partir de las larvas que tenga el criadero. Los “edificios” zerg habilitan a las larvas a transformarse en las nuevas unidades que ese “edificio” habilite. Los zerg solo pueden construir sobre su moho, el cual se expande progresivamente desde los criaderos. Cuando se construye un criadero el mismo tiene un radio de moho de 5. Cada 2 turnos suma + 1 el radio de moho. Si se destruye el criadero NO desaparece el moho.

- ***Criadero**: Estructura en donde nacen las Larvas. Empieza con 3 larvas. Máximo 3 larvas en simultáneo. Genera una larva nueva por turno sino tiene las 3.*
  - *Cada zángano puede trabajar sobre 1 mineral a la vez, recogiendo 10 unidades de mineral por turno.*
- ***Extractor**: Se construye en un géiser de Gas Vespeno para refinarlo y extraerlo. Soporta como máximo 3 zánganos quitando gas. Cada zángano extrae 10 unidades de gas por turno.*
- ***Reserva de reproducción**: Construcción que permite evolucionar larvas a zerlings Es el prerrequisito para poder construir la guarida.*
- ***Guarida**: Construcción que permite evolucionar Larvas a hidraliscos*
- ***Espiral**: Permite crear Mutaliscos. Require guarida para construirse.*

Nombre	Costo	Tiempo construcción	Vida	Unidad que habilita a construir
Criadero	50 M	4	500	• Zángano
Reserva de reproducción	150 M	12	1000	• Zerling
Extractor	100 M	6	750	-
Guarida	200 M / 100 G	12	1250	• Hidralisco
Espiral	150 M / 100 G	10	1300	• Mutalisco

Protoss

Todos los edificios protoss cuentan con un escudo y una vida. Cuando son dañados, los escudos se regeneran automáticamente hasta volver al 100 %, pero si el daño fue tal que ya les empezó a sacar parte de la vida, la misma no se recupera y queda en su estado dañado, sólo llenándose el escudo. Los protoss solo construyen sobre el radio de alcance del pilón. Un pilón tiene un radio de 3. Si el pilón es destruido los edificios que se encuentran energizados por él dejan de funcionar automáticamente (Salvo que también estén dentro del radio de otro(s) pilones en funcionamiento aún)

- ***Nexo mineral**: Recolecta mineral sobre un cristal en particular. (No requiere energía del Pilón)*
- ***Pilón**: Excavado de los Cristales Khaydarin sagrados, los Pilones Protoss actúan como puntos focales de la matriz psi emitida por el Nexo.*
- ***Asimilador**: El Asimilador prepara en cápsulas el gas producido en cualquier géiser de gas vespeno sobre el que esté construido. Suma 20 de gas por turno. (No necesita la energía del Pilón)*
- ***Acceso**: La grieta-distorsión que se encuentra en el centro de la enorme entrada es donde las tropas Protoss altamente entrenadas pueden ser transportadas desde Aiur.*
- ***Puerto estelar (requiere acceso)**: Esta estructura transporta unidades protoss en órbita de Aiur hasta los lejanos planetas en los que han de luchar por su raza.*

Nombre	Costo	Tiempo construcción	Vida	Unidad que habilita a construir
Nexo mineral	50 M	4	250 E/ 250V	-
Pilón	100 M	5	300 E/ 300 V	-
Asimilador	100 M	6	450 E / 450 V	-
Acceso	150 M	8	500 E / 500 V	• Zealot • Dragon
Puerto Estelar	150 M / 150 G	10	600 E / 600 V	• Scout

Unidades

En el juego existen unidades de tierra y de aire. Además, cada una de ellas tiene definido si solo puede atacar tierra, solo atacar aire o puede atacar a ambos objetivos.

Zerg

Nombre	Superficie	Costo	Tiempo construcción	Daño	Rango Ataque	Vida
Zerling	Tierra	25 M	2	4 T	1	35
Hidralisco	Tierra	75 M / 25 G	4	10 A / 10 T	4	80
Mutalisco *	Aire	100 M / 100 G	7	9 A / 9 T	3	120

* El mutalisco tiene una evolución posible. Esto quiero decir que una vez engendrado un mutalisco, se podrá seleccionar un mutalisco individualmente y hacerlo evolucionar a:

Nombre	Superficie	Costo de evolución	Tiempo construcción	Daño	Rango Ataque	Vida
Guardian	Aire	50 M / 100 G	4	25 T	10	100

Con lo cual por ejemplo un jugador podría tener 6 mutaliscos, elegir 2, pagar los costos y evolucionarlos a guardianes. No se puede engendrar desde el Criadero Guardianes. Se engendra un mutalisco y si se quiere se lo selecciona y se lo hace evolucionar a ese mutalisco seleccionado.

Protoss

Nombre	Superficie	Costo	Tiempo construcción	Daño	Rango Ataque	Vida E / V
Zealot	Tierra	100 M	4	8 T	1	60/100
Dragon	Tierra	125 M / 50 G	6	20 A / 20 T	4	80/100
Scout	Aire	300 M / 150 G	9	14 A / 8 T	4	100/150

Turnos

El juego es por turnos, cada jugador elige las acciones que desea hacer en su turno y luego le pasa el turno a su contrincante.

Jugadores

La aplicación se juega con 2 jugadores en simultáneo, uno vs el otro. Al iniciar el juego el sistema debe consultar a los usuarios su nombre, su color y con qué raza jugará la partida.

Validaciones:

- Nombre de jugador debe contener por lo menos 6 caracteres.
- Jugador 2 no puede tener el mismo nombre, ni color, ni raza que Jugador 1.

Comienzo de partida

Cada jugador empieza en una base[1] y con 200 de mineral. Los jugadores empiezan en lados opuestos del mapa.

Mapa

Los mapas deben tener **áreas de tierra** y **áreas espaciales**. Sólo las unidades voladoras pueden circular libremente por cualquier tipo de superficie. Los tamaños de los mapas se definen por la cantidad de bases que presenten y la distribución de las mismas deben ser equidistante al punto de origen de cada jugador. No puede haber por ejemplo un mapa que tenga 6 bases en total y haya 5 de una lado muy próximas a dónde comenzó un jugador..

[1] Se llama base a un lugar en el mapa donde encontraremos muchos cristales juntos y un volcán de gas.

4. Interfaz gráfica

La interacción entre el usuario y la aplicación deberá ser mediante una interfaz gráfica intuitiva. Consistirá en una aplicación de escritorio utilizando JavaFX y se pondrá mucho énfasis y se evaluará como parte de la consigna su usabilidad. (*en el anexo I se explicarán algunas buenas prácticas para armar la interfaz gráfica de usuario o GUI*)

5. Herramientas

1. JDK (Java Development Kit): Versión 1.8 o superior.
2. JavaFX
3. JUnit 5 y Mockito: Frameworks de pruebas unitarias para Java.
4. IDE (Entorno de desarrollo integrado): Su uso es opcional y cada integrante del grupo puede utilizar uno distinto o incluso el editor de texto que más le guste. Lo importante es que el repositorio de las entregas no contenga ningún archivo de ningún IDE y que la construcción y ejecución de la aplicación sea totalmente independiente del entorno de desarrollo. Algunos de los IDEs más populares son:
    a. Eclipse
    b. IntelliJ
    c. Netbeans
5. Herramienta de construcción: Se deberán incluir todos los archivos XML necesarios para la compilación y construcción automatizada de la aplicación. El informe deberá contener instrucciones acerca de los comandos necesarios (preferentemente también en el archivo README.md del repositorio). Puede usarse Maven o Apache Ant con Ivy.
6. Repositorio remoto: Todas las entregas deberán ser subidas a un repositorio único en GitHub para todo el grupo en donde quedarán registrados los aportes de cada miembro. El repositorio puede ser público o privado. En caso de ser privado debe agregarse al docente corrector como colaborador del repositorio.
7. Git: Herramienta de control de versiones
8. Herramienta de integración continua: Deberá estar configurada de manera tal que cada *commit* dispare la compilación, construcción y ejecución de las pruebas unitarias automáticamente. Algunas de las más populares son:
    a. Travis-CI
    b. Jenkins
    c. Circle-CI
    d. GitHub Actions (recomendado)

Se recomienda basarse en la estructura del proyecto base armado por la cátedra.

6. Entregables

Para cada entrega se deberá subir lo siguiente al repositorio:

1. Código fuente de la aplicación completa, incluyendo también: código de la prueba, archivos de recursos.
2. Script para compilación y ejecución (Ant o Maven).
3. Informe, acorde a lo especificado en este documento (en las primeras entregas se podrá incluir solamente un enlace a Overleaf o a Google Docs en donde confeccionen el informe e incluir el

archivo PDF solamente en la entrega final).

No se deberá incluir ningún archivo compilado (formato .class) ni tampoco aquellos propios de algún IDE (por ejemplo .idea). Tampoco se deberá incluir archivos de diagramas UML propios de alguna herramienta. Todos los diagramas deben ser exportados como imágenes de manera tal que sea transparente la herramienta que hayan utilizado para crearlos.

7. Formas de entrega

Habrá 5 entregas formales que tendrán una calificación de APROBADO o NO APROBADO en el momento de la entrega. Además, se contará con una entrega 0 preliminares.

Aquel grupo que acumule 2 no aprobados, quedará automáticamente desaprobado con la consiguiente pérdida de regularidad en la materia de todos los integrantes del grupo. *En cada entrega se deberá incluir el informe actualizado*.

8. Evaluación

El día de cada entrega, cada ayudante convocará a los integrantes de su grupo, solicitará el informe correspondiente e iniciará la corrección mediante una entrevista grupal. Es imprescindible la presencia de todos los integrantes del grupo el día de cada corrección.

Se evaluará el trabajo grupal y a cada integrante en forma individual. El objetivo de esto es comprender la dinámica de trabajo del equipo y los roles que ha desempeñado cada integrante del grupo. Para que el alumno apruebe el trabajo práctico debe estar aprobado en los dos aspectos: grupal e individual (se revisarán los commits de cada integrante en el repositorio).

Dentro de los ítems a chequear el ayudante evaluará aspectos formales (como ser la forma de presentación del informe), aspectos funcionales: que se resuelva el problema planteado y aspectos operativos: que el TP funcione integrado.

9. Entregables para cada fecha de entrega

Cada entrega consta de las pruebas + el código que hace pasar dichas pruebas.

Entrega 0 (Semana 11 del calendario)

- Planteo de modelo tentativo con diagramas de clases.
- Repositorio de código creado según se explica aquí.
- Servidor de integración continua configurado.
- Al menos un commit realizado por cada integrante, actualizando el README.md

Entrega 1 (Semana 12 del calendario - 10 de Noviembre)

Pruebas (sin interfaz gráfica)

Caso de uso 1

- Criadero se inicia con 3 larvas, se consume una para engendrar un zángano, le quedan 2 y después de 1 turno vuelve a tener 3 larvas. Lo mismo al consumir 2 y las 3 larvas, verificar que se regeneren acorde a los tiempos estipulados.

Caso de uso 2

- Verificar que cada edificio / construcción tarde en construirse lo que dice que tarda y que recién están “operativos” cuando ya se terminaron de construir.

Caso de uso 3

- Verificar que solo Asimilador y extractor se puedan construir sobre el gas.

Caso de uso 4

- Verificar que extractor sin zánganos trabajando no genera gas. Verificar que con 1 saca 10, con 2 20, con 3 30 y que no puede recibir a un 4to zángano porque está lleno. Verificar que el Asimilador recoge gas una vez construido según lo estipulado.

Caso de uso 5

- Verificar que no se puedan construir edificios fuera del rango de un pilon o fuera del moho.

Caso de uso 6

- Verificar el crecimiento del moho acorde a lo estipulado.

Caso de uso 7

- Verificar la recolección de minerales para ambas razas.

Caso de uso 8

- Verificar que sino se tienen los recursos no se pueden construir los edificios (Para cada edificio para cada raza).

Caso de uso 9

- Verificar que un edificio protoss sigue operativo si le destruyen un pilon que lo energiza pero aún está dentro del rango de otro que también lo energiza.

Caso de uso 10

- Verificar que al dañar una construcción zerg, la misma recupera la vida por turnos hasta volver a tener el 100%.

Caso de uso 11

- Verificar que al dañar una construcción protoss sin quitarle todo el escudo, la misma recupera su escudo por turnos hasta volver a tener el 100% del mismo

Caso de uso 12

- Verificar que al dañar una construcción protoss quitandole todo el escudo y parte de la vida la misma recupera SOLO su escudo por turnos hasta volver a tener el 100% del mismo.

Caso de uso 13

- Se destruye un criadero e igual se puede seguir construyendo sobre el moho que dejó.

Caso de uso 14

- Un pilón no puede energizar en un área ya cubierta por moho (Es decir las construcciones protoss no se pueden hacer sobre moho, por más que estén dentro del alcance de un pilón).
- El moho se puede expandir por un área no ocupada (es decir que no tenga un edificio ya construido) aunque ésta esté energizada por un pilón.

Caso de uso 15

- Verificar que no se sigan recolectando ni gas ni mineral una vez agotados los recursos del nodo mineral o del volcán.

Caso de uso 16

- Verificar que no se pueda construir sobre un volcán con una edificación ya existente (propia o del enemigo)
- Verificar que no se pueda construir un nexo mineral si hay un zángano trabajando en él y viceversa (zángano no puede extraer mineral si ya tiene un nexo construido sobre el nodo).

Caso de uso 17

- Verificar las “correlativas” de construcción.

Entrega 2 (Semana 13 del calendario)

Pruebas (sin interfaz gráfica)

Caso de uso 18

- Verificar que todas las unidades cuasen el daño que dicen que causan en sus ataques.

Caso de uso 19

- Verificar que todas las unidades que no tienen ataque de aire no puedan atacar a una unidad voladora. Idem para todas las combinaciones.

Caso de uso 20

- Verificar que solo las unidades voladoras puedan moverse por áreas espaciales.

Caso de uso 21

- Verificar que un mutalisco no pueda evolucionar a guardian sino hay recursos suficientes

Caso de uso 22

- Verificar los tiempos de construccion de unidades y que tengan el edificio que permite construirlas.

Caso de uso 23

- Verificar que un una unidad no pueda dañar a la otra o a un edificio sino está en el rango de ataque de la misma.

Caso de uso 24

- Verificar que cada jugador comienza en una base en extremos opuestos del mapa.

Caso de uso 25

- Verificar los requisitos para crear a los jugadores.

Entrega 3 (Semana 14 del calendario)

Pruebas (sin interfaz gráfica)

Modelo del juego completamente terminado (Con los detalles oportunamente especificados pasada la entrega 2)

Entrega 4 (Semana 15 del calendario)

Interfaz gráfica inicial básica: Pantallas iniciales, comienzo del juego y visualización del mapa e interfaz de usuario básica.

Entrega 5 - Final: (Semana 16 del calendario)

Trabajo Práctico completo funcionando, con interfaz gráfica final, sonidos e informe completo.

Tiempo total de desarrollo del trabajo práctico:
6 semanas

10. Informe

El informe deberá estar subdividido en las siguientes secciones:

Supuestos

Documentar todos los supuestos hechos sobre el enunciado. Asegurarse de validar con los docentes.

Diagramas de clases

Varios diagramas de clases, mostrando la relación estática entre las clases. Pueden agregar todo el texto necesario para aclarar y explicar su diseño de manera tal que el modelo logre comunicarse de manera efectiva.

Diagramas de secuencia

Varios diagramas de secuencia, mostrando la relación dinámica entre distintos objetos planteando una gran cantidad de escenarios que contemplen las secuencias más interesantes del modelo.

Diagrama de paquetes

Incluir un diagrama de paquetes UML para mostrar el acoplamiento de su trabajo.

Diagramas de estado

Incluir diagramas de estados, mostrando tanto los estados como las distintas transiciones para varias entidades del modelo.

Detalles de implementación

Deben detallar/explicar qué estrategias utilizaron para resolver todos los puntos más conflictivos del trabajo práctico. Justificar el uso de herencia vs. delegación, mencionar que principio de diseño aplicaron en qué caso y mencionar qué patrones de diseño fueron utilizados y por qué motivos.

> IMPORTANTE
>
> No describir el concepto de herencia, delegación, principio de diseño o patrón de diseño. Solo justificar su utilización.

Excepciones

Explicar las excepciones creadas, con qué fin fueron creadas y cómo y dónde se las atrapa explicando qué acciones se toman al respecto una vez capturadas.

Anexo 0: ¿Cómo empezar?

Requisitos, análisis

¿Cómo se empieza? La respuesta es lápiz y papel. No código!.

1. Entiendan el dominio del problema. Definir y utilizar un lenguaje común que todo el equipo entiende y comparte. Ej.: Si hablamos de “X entidad”, todos entienden que es algo ... Si los conceptos son ambiguos nunca podrán crear un modelo congruente.
2. Compartan entre el grupo, pidan opiniones y refinen la idea en papel antes de sentarse a programar. Pueden escribir en ayudante-virtual si tienen dudas

Anexo I: Buenas prácticas en la interfaz gráfica

El objetivo de esta sección es recopilar algunas recomendaciones en la creación de interfaces gráficas de usuario o GUI. La idea no es exigir un diseño refinado y prolijo, sino que pueda ser usado por el grupo de docentes de Algo3 sin impedimentos “lo mejor posible”.

Prototipo

Venimos escribiendo código, integrales y UML todo el cuatrimestre. Me están pidiendo una interfaz gráfica. ¿Cómo se empieza? La respuesta es lápiz y papel. No código!.

1. Armen un dibujo o prototipo en papel (pueden usar google docs o cualquier herramienta también) de todas las “pantallas”. No tiene que ser perfecto.
2. Compartan entre el grupo, pidan opiniones y refinen la idea en papel antes de sentarse a programar. Pueden escribir en ayudante-virtual si tienen dudas

JavaFX

Entender cómo funciona JavaFX es clave para implementar la GUI correctamente. No subestimen el tiempo que lleva implementar y modificar la UI o interfaz de usuario. Lean todo lo que ofrece y las buenas prácticas de la tecnología. Hay plugins específicos para el IDE, pero por más que usen herramientas WYSIWYG, siempre conviene entender la API para mejorar el código autogenerado que casi nunca es óptimo.

Recomendaciones visuales

Tamaño de elementos

- Se recomienda un tamaño de tipografía de al menos a 10 puntos al mayor contraste negro contra blanco para asegurar la legibilidad, o bien 12 puntos.
- Para los botones o elementos interactivos, el tamaño mínimo del área debería ser de 18x18.

- Extra: Los elementos más importantes deberían ser más grandes y estar en posiciones más accesibles (poner ejemplos)

Contraste

- Aseguren que el texto y los elementos tengan buen contraste y se puedan leer bien
- Se sugiere un contraste cercano a 3 entre textos y fondos para texto grande e imágenes.
- Herramientas para verificar contraste: https://colourcontrast.cc/ https://contrast-grid.eightshapes.com

Uso del color

- La recomendación es no utilizar más de 3 colores para la UI y los elementos
- En el enunciado se ejemplifica con una paleta accesible, pero pueden usar cualquiera para las fichas
- Accesibilidad: Si usan para las fichas rojo y verde, o verde y azul, aseguren que las personas con daltonismo puedan distinguirlas usando letras o símbolos sobre las mismas.
- Dudas eligiendo paletas? https://color.adobe.com

Tipografía

- No se recomienda usar más de 2 tipografías para toda la aplicación
- Asegurarse de que esas tipografías se exporten correctamente en en TP
- Evitar tipografías “artísticas” para texto, menú y botones, ya que dificultan la lectura

Recomendaciones de interacción

Manejo de errores

- No escalar excepciones a la GUI. Es un No absoluto. Enviar a la consola.
- Si muestran errores al usuario, el mensaje de error debe estar escrito sin jerga técnica y permitir al usuario continuar y entender lo que está pasando
- Siempre optar por validar y prevenir errores, a dejar que el usuario ejecute la acción y falle.

Confirmaciones

- Antes de cerrar o ejecutar cualquier operación terminal, una buena práctica es pedir confirmación al usuario (para el alcance del tp no sería necesario)

Visibilidad del estado y otros

- Mostrar el estado en el cual está el juego. Siempre debería estar accesible
- Permitir al usuario terminar o cerrar en cualquier momento