Complejidad Temporal, Estructuras de Datos y Algoritmos

Enunciado Trabajo Final

Consideraciones Generales

El objetivo de este trabajo es integrar los contenidos vistos en la materia. El trabajo deberá realizarse en forma individual.

La fecha límite para su defensa se encuentra publicada oportunamente.

Se debe crear un repositorio en **github.com** llamado ***TPF_su_apellido*** en el cual el profesor debe participar como colaborador en el mismo. El repositorio git registrará todo el proceso de codificación y la entrega del trabajo final se realizará a través de dicho repositorio.

El trabajo se presentará junto con un informe que debe incluir:

- Datos del autor del trabajo final.
- Un diagrama de clases UML describiendo la estructura del sistema, mostrando las clases del sistema, sus atributos, métodos y las relaciones entre los objetos.
- Detalles de implementación, como por ejemplo, problemas encontrados y como fueron solucionados, condiciones de ejecución, etc.
- Las imágenes de todas las pantallas que componen el sistema codificado junto con una descripción completa de las mismas.
- Ideas o sugerencias para mejorarlo o realizar una versión avanzada del mismo.
- Una breve conclusión o reflexión de la experiencia adquirida a partir de la realización del trabajo final.

Este informe se evaluará tanto en su contenido como en la forma en el que es presentado y tendrá una nota que afectará la nota final del trabajo.

El trabajo deberá defenderse en un coloquio presencial.

Enunciado

Una empresa informática está llevando a cabo un juego de computador que se desarrolla como una conquista planetaria donde el enfrentamiento se da únicamente entre dos jugadores. Estos últimos están representados por el usuario y por una entidad de software que posee codificada cierta estrategia destinada a ganar el juego (**Bot**). El objetivo del juego es apoderarse de todos los planetas del rival. Es un juego basado en turnos, en donde los participantes disponen de un número de turnos determinado para conseguir la victoria. En cada turno, los planetas que se muestran en pantalla, albergan una cantidad específica de naves que pueden pertenecer al jugador, al oponente o ser neutrales. La propiedad está representada por un color asignado a cada jugador, a saber: rojo para el usuario, azul para el **Bot** y blanco para indicar neutralidad. Además, cada planeta tiene un conjunto de rutas interplanetarias que los interconecta y que sirven para enviar flotas de un planeta hacia otro. Además los planetas cuentan con una tasa de crecimiento que indica cuantas naves pueden generar durante cada asalto, que luego serán agregadas a la flota que el jugador posee en el planeta. Los planetas neutrales no agregan nuevas naves a la flota que alojan hasta que sea conquistado por algún jugador.

Cada jugador se implementa como un agente de software que, a partir de una lista de planetas y sus flotas (el estado actual del mapa), devuelve una acción a realizar. En cada turno, se debe elegir el movimiento que realizará una flota que partirá de un planeta propio a cualquier otro planeta destino, pudiendo ser este último propio o no. Los movimientos serán posibles siempre y cuando exista una ruta interplanetaria directa entre un origen y un destino. Las flotas, pueden necesitar más de un turno para llegar a su planeta de destino (el tiempo es directamente proporcional a la distancia entre los planetas). Cuando la flota llega un planeta enemigo o neutral, tiene lugar un enfrentamiento, en el cual cada nave es sacrificada para destruir una nave enemiga, en otras palabras, resulta victoriosa aquella flota que albergue más naves. En caso de que el planeta de destino sea del propio jugador, ambas flotas se unen, sumando sus naves. En cada turno, el número de naves alojadas en los planetas de los jugadores (no los neutros), se incrementa de acuerdo a la tasa decrecimiento de cada planeta.

La empresa informática lo ha contratado a Ud. para implementar la estrategia de ataque que utiliza el **Bot** cuando en el mapa se disponen los planetas jerárquicamente o en forma de **árbol general**. Esta estrategia tiene como objetivo conquistar el planeta ubicado en la raíz del árbol para luego atacar el planeta enemigo más cercano hasta que no haya más planetas enemigos por atacar. En el juego la codificación de la estrategia antes descripta se realizará en la clase **Estrategia**.

Los métodos de la clase **Estrategia** que Ud. debe implementar son:

1. **CalcularMovimiento** (ArbolGeneral<Planeta> arbol**)**: Este método calcula y retorna el movimiento apropiado según el estado del juego. El estado del juego es recibido en el parámetro **arbol** de tipo **ArbolGeneral<Planeta>**. Cada nodo del árbol contiene como dato un planeta del juego.

2. **Consulta1** (ArbolGeneral<Planeta> arbol): Calcula y retorna un texto con la distancia que existe entre la raíz y el nodo del árbol que es enviado como parámetro que contiene el planeta más cercano perteneciente al **Bot**.

3. **Consulta2** (ArbolGeneral<Planeta> arbol): Calcula y retorna en un texto la cantidad de planetas que tienen población mayor a 10 en cada nivel del árbol que es enviado como parámetro.

4. **Consulta3** (ArbolGeneral<Planeta> arbol): Calcula y retorna en un texto el promedio poblacional por nivel del árbol que es enviado como parámetro.

Las clases con la implementación del juego provista por la catedra a fin de simplificar la construcción del mismo, deben obtenerse con el comando de git:

```
git clone https://github.com/petinato54/cteda2020_TPF.git
```

A continuación se detallan los métodos de las clases **Planeta** y **Movimiento** que Ud. necesitará para desarrollar lo solicitado:

Clase Planeta
– bool **EsPlanetaDeLaIA**(): Si el planeta pertenece al **Bot** retorna **true** y **false** en caso contrario.
– bool **EsPlanetaDelJugador**() : Si el planeta pertenece al usuario del juego retorna **true** y **false** en caso contrario.
– bool **EsPlanetaNeutral**(): Si el planeta no pertenece ni al usuario del juego, ni al **Bot** retorna **true** y **false** en caso contrario.
– int **Poblacion**(): Retorna la población actual del planeta.

Clase Movimiento
– **Movimiento** (Planeta o, Planeta d): constructor que toma como parámetro el planeta origen y destino del movimiento.
– Planeta **origen** { get; set; }: getter y setter del planeta origen del movimiento.
– Planeta **destino** { get; set; }: getter y setter del planeta destino del movimiento.

Aclaración : La clase Juego implementa el método main que inicia el juego.