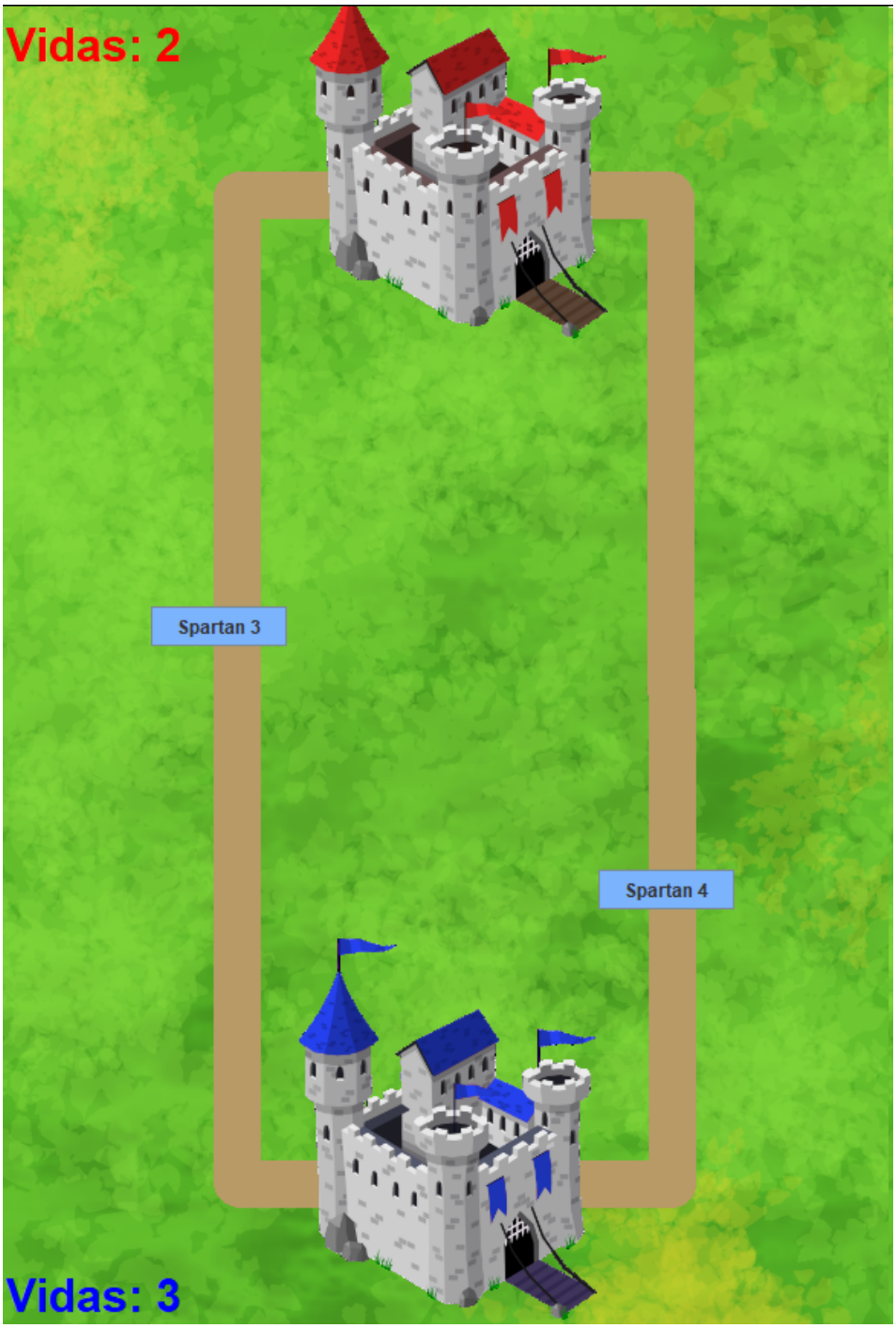
Taller de Lenguajes 2

Trabajo Final Integrador

Tower Defense



🎯 Objetivo General

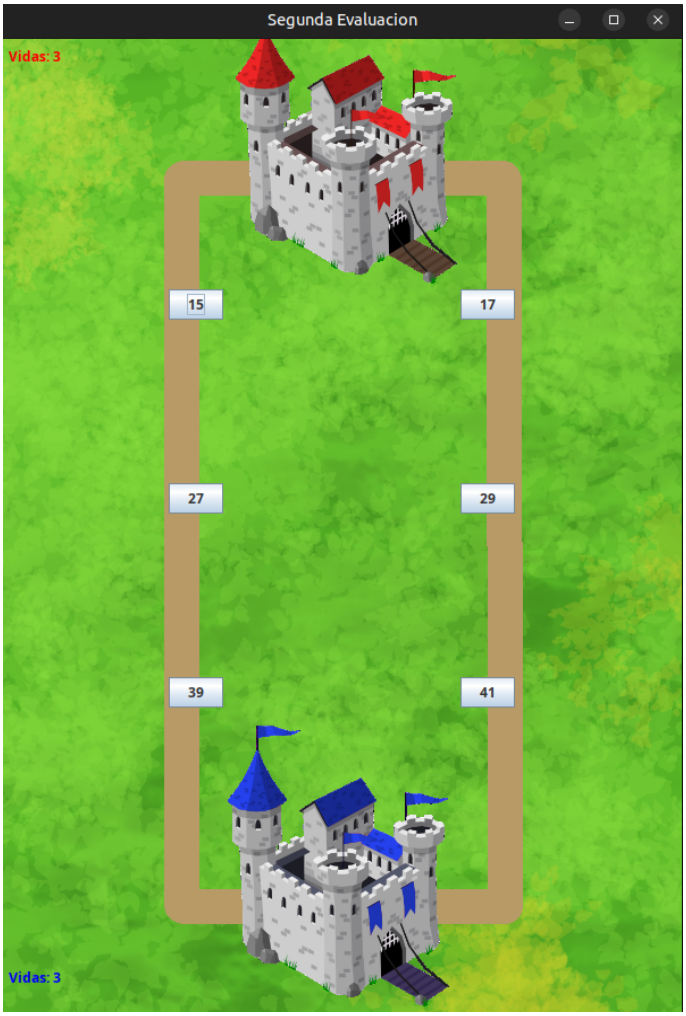
Modificar el código del proyecto **entregaFinal** agregando nueva funcionalidad.
El proyecto implementa el juego “Tower Defense” de 2 jugadores, en el que participan 3 clases de **monstruos** (Spartan, EvilBeast y IceBeast) que pueden pertenecer a algunos de los 5 **tipos** (BEAST, COLD, FIRE, SWORD, DEMON) y que pueden desplegar 4 tipos de **ataques** (Slice, ColdBreath, Curse, IceSpike). El objetivo del juego es competir contra el enemigo y derrotarlo.

El **objetivo de este trabajo integrador** consiste en **agregar** al juego **nuevos monstruos, tipos de monstruos y ataques**. Esperamos ver en sus trabajos cosas variadas, entretenidas y divertidas!!!.

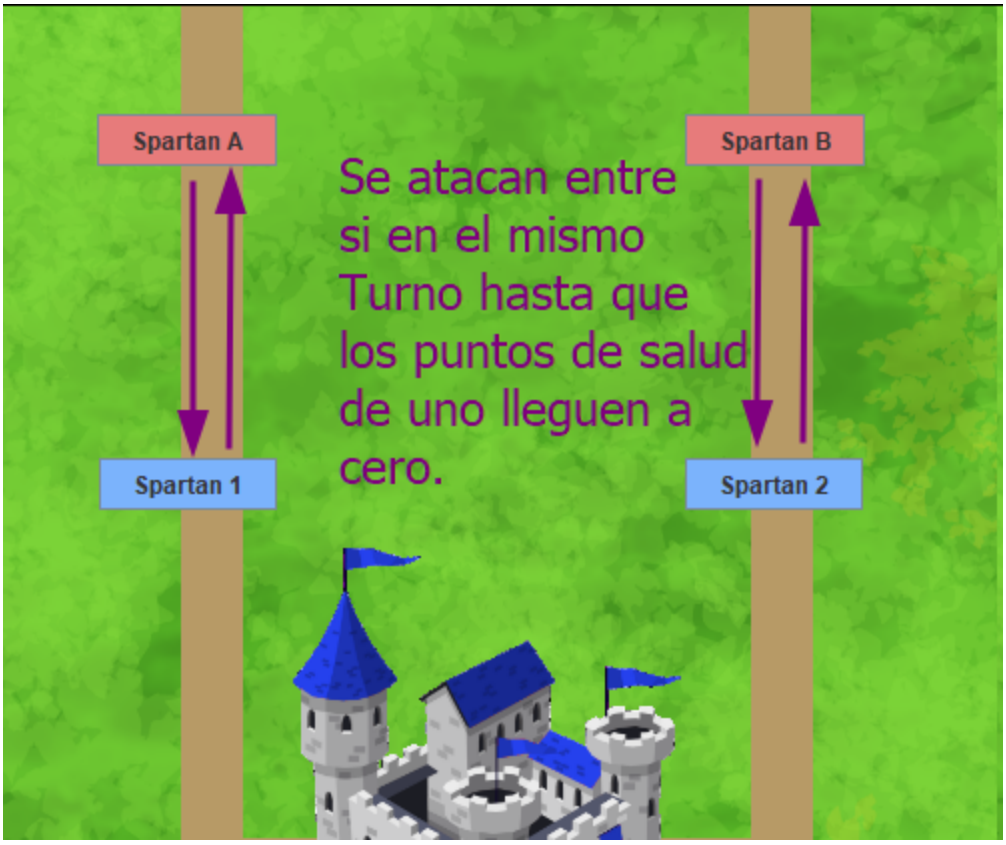
💬 Descripción general del juego Tower Defense

El Juego consiste de **2 jugadores**, Rojo y Azul, cada uno de ellos defiende la torre del color correspondiente. A su vez, disponen de una **lista de monstruos** que va a ir desplegando diferentes **ataques** a lo largo de la partida para defender su torre y atacar la del enemigo. Datos importantes:

- Existen 2 líneas, Oeste y Este.
- Cada jugador dispone de 3 vidas.
- No puede haber más de un monstruo por línea de cada jugador.
- El juego es por rondas, una ronda ocurre cada 1.5 segundos.
- En las rondas los monstruos se mueven de un casillero a otro, atacan o entran en el castillo enemigo.
- El orden de las rondas puede variar: el jugador azul o rojo podrían atacar 2 veces seguidas esto hace que 2 partidas con la misma formación de monstruos pueda tener resultados distintos.
- Cada línea tiene 3 casillas que separan un castillo del otro, como se muestra en la siguiente figura:



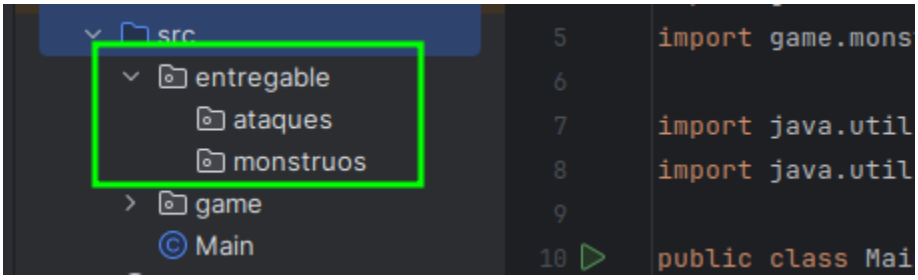
- Los monstruos usan las casillas para moverse de un castillo a otro, por ejemplo un monstruo puede comenzar en la casilla número 39, en su siguiente turno avanzar hasta 27, luego hacia la 15 y finalmente ingresar al castillo del jugador rojo para descontarle un punto de vida.
- Cuando un monstruo llega al castillo del enemigo, este último pierde una de las 3 vidas.
- Una vez que las vidas llegan a 0 el jugador pierde.
- Cuando un monstruo quiere ocupar una casilla ocupada por otro monstruo del jugador contrario, estos pelean atacando un monstruo al otro y viceversa, como se muestra en la siguiente figura:



- Los monstruos tienen puntos de salud que se decrementan con cada ataque, cuando este valor llega a 0 el monstruo es eliminado y retirado de la línea.

Descripción del proyecto JAVA

La funcionalidad del proyecto está organizada en 2 paquetes: **entregable** y **game.** En el **paquete game** se encuentran las clases que componen el juego original y el **paquete entregable** es en el cual se debe agregar el código nuevo.



Los jugadores tienen un listado de monstruos, la creación y asignación de esos monstruos la pueden ver en el archivo **Main,** un ejemplo de un monstruo puede ser el **Spartan** el cual tiene

una propiedad **Type** y otra **activeSkill.** Los monstruos pueden pertenecer uno o más tipos, los cuáles determinan sus habilidades, por ejemplo el tipo Fuego es más fuerte que el Hielo y más débil que el Metal o la Roca.

A continuación se muestra la clase abstracta Monster, la cual es obligatorio extender para crear un nuevo monstruo y equiparlo con una estrategia concreta de ataque (método attack). Por otro lado, la clase Monster implementa 2 métodos extras que son importantes: **move()** y **onDamageReceive().** Aunque Monster ofrece una implementación predeterminada, estos métodos pueden sobreescribirse para proveer funcionalidad personalizada.

```
public abstract class Monster {

   public abstract void attack(Monster monster);

   public void onDamageReceive(Integer damage, Monster monster) {
       this.life = this.life - damage;
       if(this.life < 0) {
           this.life = 0;
       }
       System.out.println(this + " fue herido, queda con " + this.life + " puntos
de vida");
   }

   public void move(PathBox oldPathBox, PathBox newPathBox) {
       oldPathBox.setMonster(null);
       newPathBox.setMonster(this);
   }
}
```

En el ejemplo siguiente, la clase **Spartan** extiende de la clase **Monster**. Es importante que desde **attack()** se invoque a **onDamageReceive()** una vez calculado el daño, como se muestra en el ejemplo, en caso contrario, no habrá efectos en el **monstruo** enemigo.
A la vez, es obligatorio **configurar en el constructor del monstruo**: los **puntos de salud** por medio de la variable **life** y el nombre del monstruo por medio de **monsterName** el cual se usará para mostrar en la UI. En este ejemplo **activeSkill** es solo una (Slice) sin embargo es posible implementar más de una; por ejemplo, **IceBeast** es un monstruo que tiene 2 habilidades (activeSkill) que intercambia cada vez que avanza un casillero.

```
public class Spartan extends Monster {

   public Spartan(String name) {
       this.life = .500;
       this.activeSkill = new Slice();
       this.monsterName = name;
       this.types = Arrays.asList(Type.SWORD);
   }

   @Override
   public void attack(Monster enemy) {
       int damage = this.activeSkill.damage(enemy);
       System.out.println("--     ["+ this +"] ataca a [" + enemy + "] haciendole "
+ damage + " de daño");
       enemy.onDamageReceive(damage, this);
   }
}
```

Las habilidades (activeSkill) implementan la interface **Attack.** La cual tiene un solo método que es **damage()** que recibe **Monster** como parámetro, que puede o no ser usado para determinar el tipo del monstruo y en base a eso hacer más o menos daño.

```
public interface Attack {
   int damage(Monster monster);
}
```

En el siguiente ejemplo, **ColdBreath** hace doble daño si el enemigo es del tipo **Fire.** En este caso **Cold** es una subinterface de Attack**.**

```
public class ColdBreath implements Cold {

   @Override
   public int damage(Monster monster) {
       int damage = RandomGenerator.getInstance().calculateDamage(90, 150);
       if(monster.getTypes().contains(Type.FIRE)) {
           damage = damage * 2;
       }
       return damage;
   }
}
```

📥 Descarga del Código

El código que se debe modificar en esta evaluación debe ser descargado desde este enlace.

📚 Descripción de tareas a realizar

Los puntos del código que debe modificarse en el proyecto JAVA están marcados con un comentario etiquetado con TODO.

- Todos los grupos **deben implementar en sus juegos:**

- Agregar 2 nuevos monstruos y sobreescribir, en al menos uno de ellos, los métodos **move()** y **onDamageReceive()** .
- Al menos 5 nuevas habilidades.
- Agregar al menos 5 nuevos tipos de monstruos y relacionarlos con los monstruos creados.

- Para **grupos de 2 y 3 integrantes**:

En la versión original, el orden de salida de los monstruos está dado por el orden de la lista: el primero de la lista es el primero en salir a la batalla.
Los grupos de 2 y 3 integrantes deben implementar al menos 2 criterios de ordenamiento para la lista de monstruos, por ejemplo por vida (el que más vida tiene sale primero a la batalla), por ataque (el que más poder de daño tiene sale primero) o por tipo**.**

- Para **grupos de 3 integrantes**:

La versión original finaliza el juego cuando las vidas de uno de los jugadores llegan a cero cuando la cantidad de rondas llega a 100.
Los grupos de 3 integrantes deben implementar:
- Una excepción en situación de “empate”: la excepción se dispara cuando ambos jugadores no tienen monstruos disponibles para atacar pero ninguno perdió.
- Implementar una ventana popup que informe el resultado del juego.

🗓️ Fecha importante y modalidad de evaluación

Lunes 18/12, a las 18:30 hs., en el aula 10A.
Cada grupo deberá subir su proyecto a una tarea del aula virtual antes del horario de la presentación del proyecto.
A cada grupo se le asignará un horario que se informará oportunamente.

La **evaluación** consistirá en:

Mostrar el trabajo funcionando.
Explicar las decisiones tomadas.
Responder preguntas sobre los conceptos teóricos usados.

🧠 Temas a evaluar

- Clases
- Herencia
- Polimorfismo
- Swing
- Excepciones
- Interfaces
- Tipos Enumerativos