**Desarrollo Teórico**

Para el desarrollo del juego “Human vs. Zombies”, se buscó aplicar los diferentes conceptos aprendidos del paradigma orientado a objetos, en base a lo que se quería diseñar: un juego inspirado en el conocido “Plants vs. Zombies”.

Se enumeran los conceptos fundamentales del Paradigma orientado a objetos con sus respectivas aplicaciones en nuestro juego:

* **Polimorfismo:**

Los mensajes polimórficos visibles en el juego son chocasteConEnemigo(unEnemigo),chocasteConBorde() y chocasteConJugador(). Los objetos que lo implementan son todos los personajes o elementos del juego que interactúan entre sí: el jugador principal, todos los enemigos, el borde definido como forma de perder el juego y los objetos especiales que el jugador puede recoger. Cuando reciben ese mensaje, cada uno podrá tener un comportamiento diferente en consecuencia, pero deben entenderlo de todas formas. El objeto que se aprovecha de este concepto es game, ya que cuando cada uno de los visuales mencionados realiza su configuración inicial, recibe el mensaje onCollideDo() con alguno de estos mensajes como parámetro para que se produzca el efecto esperado en estos objetos al producirse la colisión.

* **Colecciones:**

Se utilizan colecciones para modelar los corazones que representan la vida del personaje, los proyectiles que la munición del personaje tiene disponible para dispararles a los enemigos y los bordes utilizados para cubrir el lado izquierdo de la pantalla. Por otro lado, también se emplean para representar las diferentes oleadas de enemigos que ocurren en cada nivel.

Los mensajes enviados a las colecciones mencionadas son:

* + **size():** en el caso de las vidas y proyectiles disponibles, es importante conocer el tamaño de las colecciones ya que el juego debe terminar si el jugador no tiene más vidas, y no se le debería permitir disparar si ya no posee más proyectiles en la colección. Este mensaje no posee efecto.
  + **add():** este mensaje sí tiene efecto, y se utiliza cuando el jugador recarga su cargador (es decir, la colección de las mismas vuelve a tener cinco instancias de la clase Proyectil) y, a su vez, cuando colisiona con el objeto curita y debe recuperar sus cinco vidas. Es enviado, también, a la colección de bordes en el momento en el que el objeto generaciónDeBordes se debe encargar de producirlos.
  + **remove():** al contrario del mensaje anterior, este se emplea cada vez que el jugador pierde una vida, o dispara y se debe disminuir la cantidad de proyectiles que el cargador del personaje posee. Por lo tanto, posee efecto sobre las colecciones.
  + **forEach():** este mensaje se emplea sobre las vidas, los proyectiles y las oleadas de enemigos para que cada uno de los objetos contenidos en las mismas reciban un mismo mensaje y se produzca un efecto en ellos. En todos los casos este es agregarse(), que muestra al objeto en la pantalla y, en el caso de las oleadas, configura el efecto de la colisión con cada uno de los enemigos en esa colección.
* **Clases:**

En lo que respecta a los elementos del juego, utilizamos la clase ElementoEspecial para establecer los atributos y métodos principales de cualquier objeto que el jugador puede recoger del suelo, como lo son el objeto curita, la mina y las balas de tipo “chetada” e “infinito”.

Además, implementamos la clase Enemigo y la clase EspecieEnemigo para definir la estructura de los contrincantes con los que se enfrenta nuestro personaje principal en cada uno de los niveles. La clase EspecieEnemigo permite crear nuevas instancias de enemigos pero a los que se le puede variar la velocidad y la cantidad de vida que poseen. Se crean nuevas instancias de esta clase a medida que el juego avanza, en lo mencionado previamente como “oleada”.

Finalmente, se tienen las clases Corazón, Proyectil, TipoProyectil, ProyectilEnemigo y Borde. La primera es utilizada para instanciar cada uno de los corazones que representan la vida de nuestro personaje. La segunda, por su parte, nos permitirá crear los proyectiles que utilizará el mismo en su cargador para defenderse contra los enemigos. TipoProyectil es la clase que permite instanciar proyectiles que varían en cuanto a imagen, el daño que provocan, la velocidad con la que se los dispara, el tiempo que tardan en recargarse.

Por último, la clase Borde permite crear las instancias necesarias para que el borde izquierdo de la pantalla quede totalmente cubierto y que, cuando un enemigo lo cruza, el juego se dé por finalizado.

* **Herencia:**

Retomando lo anteriormente explicado de los elementos de juego, los objetos curita, mina y las balas chetada e infinito, heredan los atributos y métodos de la clase ElementoEspecial, tales como chocasteConEnemigo(unEnemigo) y agregarse() de la superclase, además de atributos que se requieren como la imagen, el tiempo hasta la aparición y el tiempo que permanece en pantalla. No obstante, ambos operan de forma distinta cuando reciben el mensaje chocasteConJugador(), y es por ello que éste es definido en forma individual.

Por su parte, el objeto boss hereda de la clase Enemigo debido a que si bien comparte atributos y métodos con cualquier otro tipo de enemigo, tiene un método único que es dispararAlPersonaje()*,* en el cual se configuran los disparos que hace el enemigo Boss. En este sentido, fue necesaria una redefinición del métododisminuirVida(cantidad)(propia de la clase Enemigo) ya que cuando el personaje logra eliminar al boss, el juego debe remover el evento que dispara.

* **Delegación de responsabilidades:**

La delegación de responsabilidades en el juego se da en los siguientes casos:

* El objeto curita tiene un método chocasteConJugador(), en el cual se delega la responsabilidad al jugador de que este mismo sea quien aumente toda su vida, mediante el mensaje aumentarTodaLaVida(), que lo entiende solamente el personaje.
* Análogamente, la clase Proyectil implementa un método denominadochocasteConEnemigo(elEnemigo), mediante el cual se le delega la responsabilidad al enemigo con el cual colisionó de disminuir su vida, mediante el método disminuirVida(cantidad).
* El objeto juego tiene un método iniciar(), en el cual se realiza la configuración básica del juego (como el título y tamaño del tablero). Dado que cuenta con tres niveles, al enviarle el mensaje iniciar()al juego delega la responsabilidad al objeto nivel\_0 para que realice su configuración: es decir, que se muestre el nombre del juego y la posibilidad de hacer clic en la tecla “e” para empezar a jugar.
* Similar a lo que sucede entre el juego y el nivel\_0, cuando se cambia de nivel, cada nivel anterior le debe delegar a la responsabilidad al siguiente para que se realice la configuración inicial del mismo. Esto se hace enviándole el mensaje configuracionInicial()*,* en el cual se agrega el fondo, se establecen las oleadas de enemigos correspondientes al nivel
* Por último, otro claro ejemplo es la creación del objeto munición. El mismo fue añadido con el fin de ser el que administre todo lo relacionado a las balas y la funcionalidad de los disparos de nuestro personaje principal. La lógica detrás del mismo, consiste en comprender que el personaje tiene una determinada munición, que se agota y la recarga, pero **no es él** el encargado de poner una por una sus balas (lo cual realiza el métodorecargar()del objeto munición) ni quitarlas cuando él dispara (realizado por el mensaje retirarUnaBala).
* **Diagrama:**

En el siguiente diagrama estático se pueden apreciar todas las clases y objetos involucrados en cualquiera de los niveles y cómo éstos interactúan.

