**Desarrollo Teórico**

Para el desarrollo del juego “Player vs Enemies”, se buscó aplicar los diferentes conceptos aprendidos del paradigma, 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, 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.

* **Colecciones:**

Se utilizan colecciones para modelar los corazones que representan la vida del personaje, los proyectiles que tiene disponibles para dispararle al enemigo 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 un nivel.

Los mensajes enviados a las colecciones mencionadas son:

* + **size():** en el caso de las vidas y bolas de fuego 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 sus bolas de fuego (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 disminuye la cantidad de bolas de fuego que posee. Por lo tanto, posee efecto sobre las colecciones.
  + **forEach():** este mensaje se emplea sobre las vidas, las bolas de fuego 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 si, es necesario, configura el efecto de la colisión con el mismo.
* **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 y el reloj de arena.

Además, implementamos la clase Enemigo para definir la estructura de los contrincantes con los que se enfrenta nuestro personaje principal. En cada nivel, el personaje principal se enfrentará a distintos enemigos, los cuales entenderán distintos mensajes. 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 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 las bolas de fuego que utilizará el mismo para defenderse contra los enemigos. Por su parte, la clase Borde permite crear las instancias necesarias para que el borde izquierdo de la pantalla quede 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 y reloj de arena, heredan los atributos y métodos de la clase ElementoEspecial. Ambos realizan su aparición unos segundos luego de que el juego ha comenzado, y heredan métodos tales como *chocasteConEnemigo(unEnemigo)* y *agregarse()* de la superclase, además de atributos que se requieren como la imagen y la posición. No obstante, ambos operan de forma distinta cuando reciben el mensaje *chocasteConJugador()*, y es por ello que éste es definido cuando se los define individualmente.

* **Delegación de responsabilidades:**

La delegación de responsabilidades en el juego se dan 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 denominado*chocasteConEnemigo(elEnemigo)*, mediante el cual se le delega la responsabilidad al enemigo con el cual colisionó de disminuir su vida, mediante el método *disminuirVida()*.
* El objeto juego tiene un método *iniciar()*, en el cual se realiza la configuración inicial del juego. Asimismo, dado que el juego presenta distintos niveles con distintas configuraciones, al enviarle el mensaje *iniciar()*al juego, éste delega la responsabilidad al primer nivel para que realice su configuración y el usuario pueda dar por comenzado el juego. Esto lo hace a partir de enviarle el mensaje *configuraciónInicial()*, que lo entiende la clase Nivel. Esta configuración del nivel incluye las listas de enemigos, el fondo, la música y de los distintos elementos que están presentes en el mismo.