La creación del videojuego fue estructurada teniendo como fundamentos principios de programación orientada a objetos, por lo que se buscó una adecuada separación de responsabilidades, reutilización de código y facilidad para extender la lógica de los niveles y de los personajes. A continuación, se justifica el uso de cada clase y su relación con la arquitectura general del sistema.

1. Clase Juego

La clase Juego representa el controlador del flujo de ejecución. Se ocupa de realizar el paso de niveles, mostrar los menús y reiniciar el estado del juego. Esta clase permite la separación de la lógica general del ciclo de juego, respecto a la lógica particular de cada nivel.

2. Clase Nivel

Cada instancia de Nivel representa un nivel diferente del videojuego. Contiene referencias a los elementos activos del escenario: jugador (Goku), enemigos (Enemigo\*), ítems (Item\*) y proyectiles (Proyectil\*). Esta clase centraliza la lógica de actualización y render del nivel, además de la lógica del paso de nivel del jugador.

3. Clase Mapa

A pesar de simplificada, la clase Mapa representa el entorno gráfico y estructural del escenario, como las plataformas, límites y trampas. Su utilización permite encapsular la lógica de colisiones y distribución de los elementos a mostrar.  
  
4. Clase abstracta Entidad

Entidad es la clase base común para cualquier objeto que tenga posición y vida dentro del juego. Al definir atributos compartidos, como posicionX, posicionY y vida, así como los métodos mover() y recibirDanio(), da la posibilidad de reutilizar la lógica entre los personajes, los enemigos, los proyectiles y el resto de los elementos interactivos.

5. Clase abstracta Personaje

Hereda de la clase Entidad y proporciona el método atacar(), común tanto a Goku como a los enemigos. Dicha abstracción es necesaria para poder manejar cualquier personaje de forma genérica y extender funcionalidades particulares en las subclases.

6. Clase Goku

Es la implementación concreta del personaje manejado por el jugador, donde hay habilidades especiales como saltar(), planear() o lanzarKamehameha(), con lo cual se hace diferente del resto de los personajes. Conceptualmente está alineada con la jerarquía al implementar el comportamiento del personaje y facilita la integración con el resto del sistema.

7. Clases Enemigo, EnemigoVolador y TenShinHan

Los enemigos heredan Personaje, ya que encapsulan el comportamiento autónomo de los oponentes del juego. EnemigoVolador aplica un movimiento oscilatorio en patrón seno y TenShinHan tiene la capacidad de lanzar proyectiles (Dodon Ray), aparece en el segundo nivel. Esta jerarquía permite incluir enemigos nuevos con un comportamiento diferente.

8. Clase Proyectil

Representan los ataques a distancia como el Kamehameha o el Dodon Ray. Dado que están en relación con algún Personaje (el que los lanza) y dado que tienen métodos de colisión y movimiento, permiten resolver el combate entre entidades sin mezclar lógicas de los personajes que no son necesarias.

9. Clase Item

Sirve para modelar objetos recolectables como pueden ser esferas del dragón o platos de arroz. Su método activar() interaccionan con Goku, modificándole atributos según el tipo de ítem que haya recolectado. Su inclusión a la arquitectura permite ampliar la jugabilidad con nuevos ítems sin alterar la parte central del jugador.

10. Clase HUD

Se encarga de mostrarnos en pantalla la vida de Goku y el número de esferas recogidas. Ese componente interacciona con el estado del jugador pero no tiene nada que ver en la lógica del juego, es decir, cumple el principio correspondiente a la separación entre lógica y presentación.

11. Relaciones entre clases

Nivel tiene dentro de él a Goku, enemigos, items, y proyectiles, con lo que se puede aglutinar toda la lógica de juego en una única unidad de reutilización.

Juego tiene bajo-control la progresión del jugador así como el intercambio de niveles.

Proyectil mantiene una referencia a su dueño para poder distinguir entre ataques de aliados y ataques de enemigos.

HUD observa a Goku para poder refrescar con rapidez la interfaz gráfica.

La conclusión refleja un diseño que satisface las propiedades del mantenimiento, el incremento del propio juego a nuevos niveles y enemigos y el cumplimiento de los requerimientos del presente curso como herencia propia, estructuras dinámicas (QVector), uso de físicas propias, y separación entre lógica y GUI.