INFORME

En esta expansión hemos seguido con el plan del primer sprint, hemos dividido las cosas en 2 carpetas principales, la carpeta relacionada con todas las clases del vista y otra carpeta con las del modelo, dentro del modelo hay una subcarpeta para las evoluciones pero nada relevante. No hay ninguna carpeta para el controlador porque va incluido en las clases del vista.

En la carpeta packVista tenemos todo lo relacionado con las interfaces, tenemos la pantalla de inicio que es la encargada de mostrar las puntuaciones y el botón de jugar, a parte de eso también tenemos la pantalla principal, que es donde se juega la partida. Para los minijuegos tenemos la clase PanelMinijuego, para TamaDigOut tenemos el PantallaTamaDigOut, y para nuestro minijuego propio, en este caso el tres en raya, tenemos PanelTresEnRaya y PantallaTresEnRaya.

Siguiendo el patrón MVC, una vista no se puede comunicar directamente con el modelo, por eso las vistas tienen sus controladores, por ejemplo en la pantalla de inicio tenemos el botón jugar, que llama a la clase del modelo Juego y ejecuta iniciarPartida().

En la carpeta del modelo incluimos todas las clases relacionadas con el modelo, por ejemplo todo lo relacionado con la partida, el tamagotchi, el control de los minijuegos e incluso del juego completo. También incluimos las clases que controlan el patrón Factory y el State, que se explicará posteriormente.

Para notificar a la vista los cambios del modelo utilizamos el patrón observer. Este patrón utiliza los métodos setChanged y notifyObserver. Por ejemplo en ListaJugadores llamamos a notifyObserver para enviarle las puntuaciones de los jugadores, mientras que la clase PantallaPrincipal recibe los datos con el update los coloca en la pantalla.

Para la gestión de las evoluciones hemos utilizado el patrón state y para los bloques de los minijuegos el patrón factory

El patrón factory lo hemos utilizado para generar los bloques del primer minijuego, de esta manera es más fácil ampliar el minijuego en un futuro.

Y por último hemos decidido utilizar el patrón state porque es un patrón de comportamiento, que es lo que necesitamos, y no hemos utilizado el strategy porque no tiene sentido utilizarlo. Y también porque la evolución del tamagotchi es independiente del usuario.