Proyecto : Juego Buscaminas

Arquitectura de software

Integrantes:

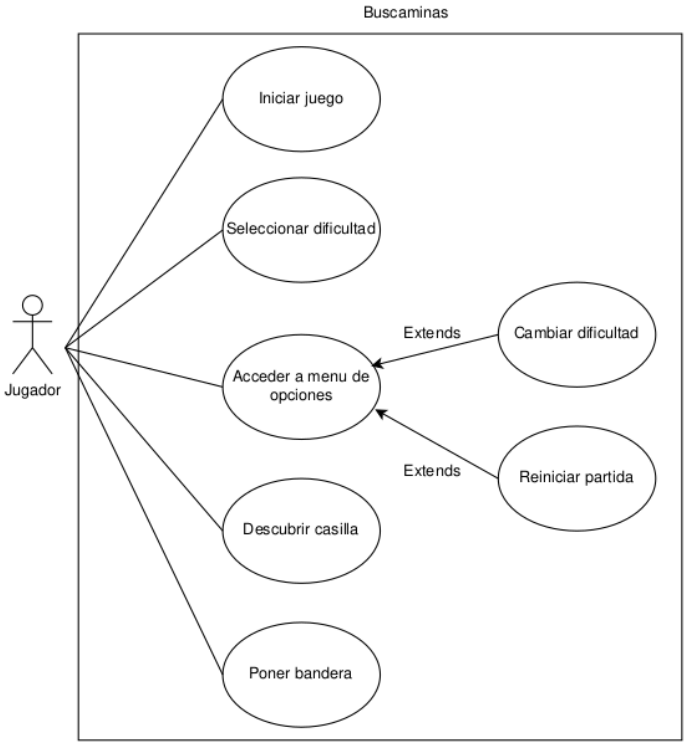
- Matias Relauquen
- Rodrigo Vargas

Introducción

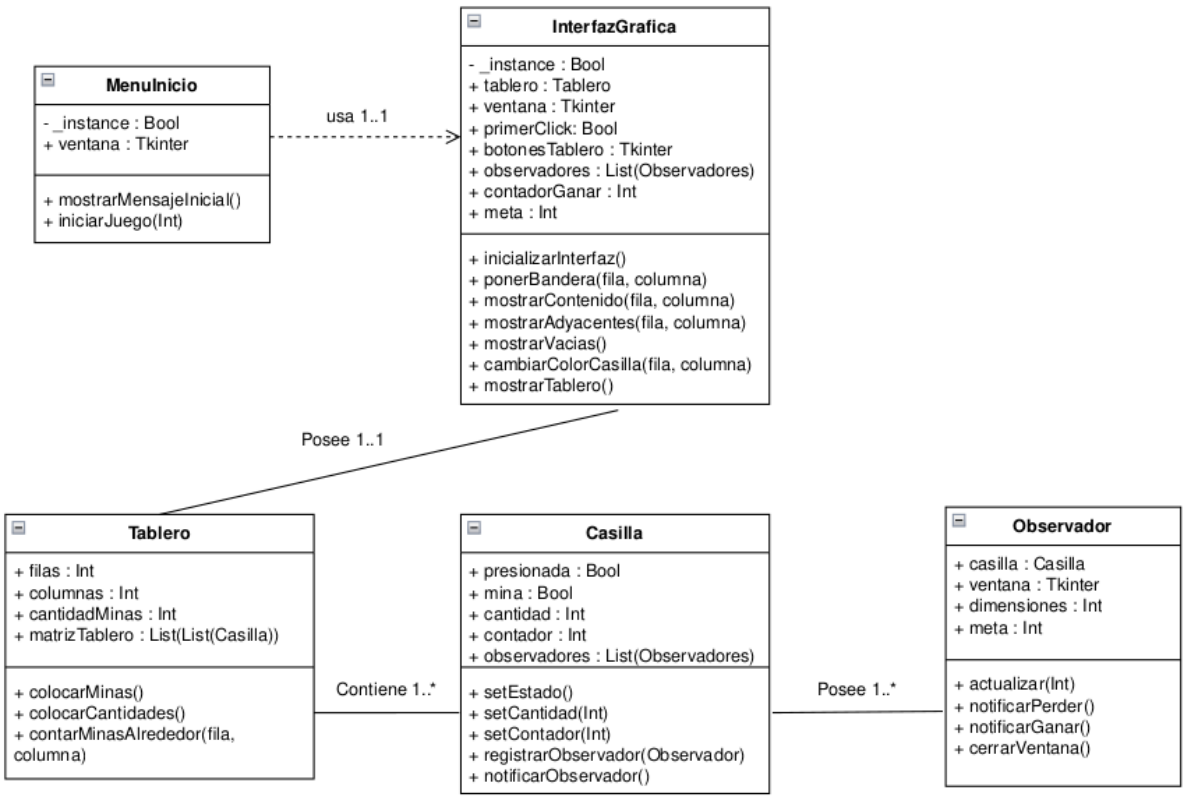
En el marco de este informe, se abordará el desafío de implementar una arquitectura de software sólida y eficiente mediante la aplicación de la metodología 4+1. Nuestro objetivo principal es garantizar la calidad del software, para lo cual nos embarcaremos en el diseño detallado siguiendo las buenas prácticas establecidas en el campo de la ingeniería de software. La implementación se llevará a cabo en el lenguaje Python, aprovechando las herramientas y facilidades que ofrece. Además, consideraremos la aplicación de patrones de diseño cuando sea pertinente, con el fin de optimizar la estructura del software.

Descripción del modelo de software: Metodologia 4+1

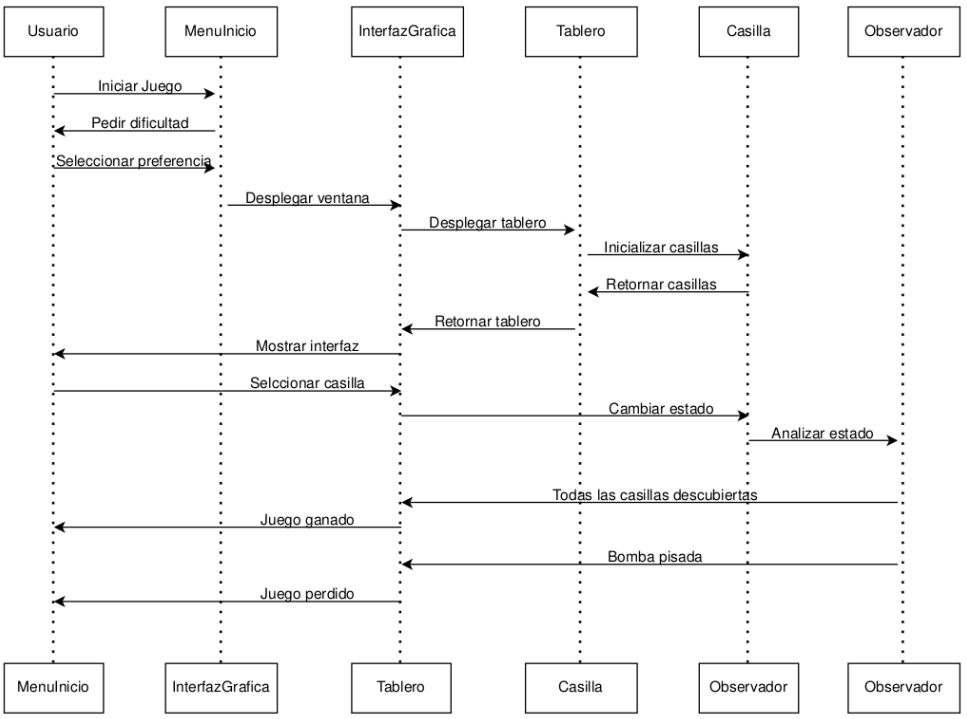
1.Diagrama de casos de uso



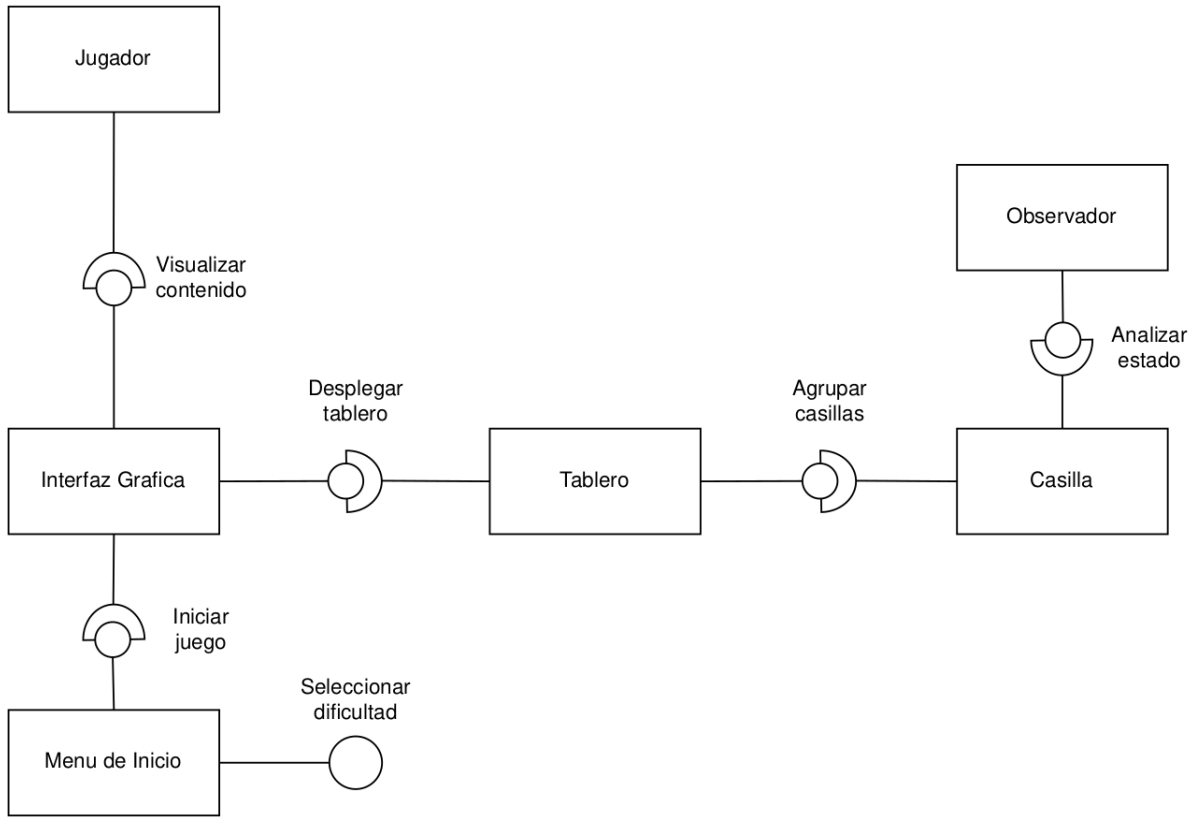
2.Diagrama de clases



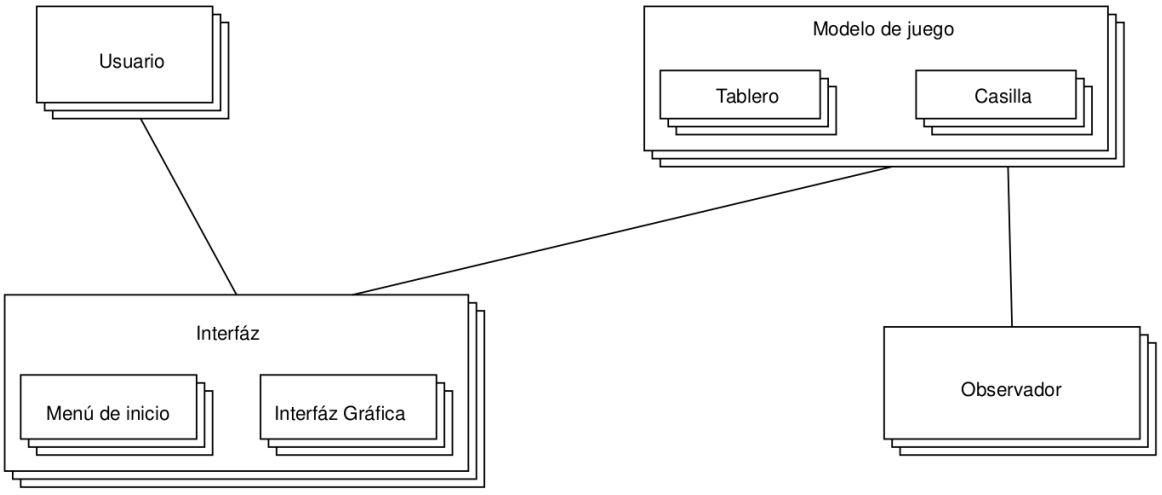
3.Diagrama de secuencia



4.Diagrama de componentes



5.Diagrama de despliegue



Patrones de diseño utilizados

Patrón de diseño	Uso en el código	Razón de uso
Singleton	MenuInicio, InterfazGrafica	Garantizar que exista solo una instancia de estas clases y entregar al jugador un único punto de acceso a estas. Los motivos de su uso son: 1. Consistencia: Ayudar a mantener la coherencia de los distintos estados dentro del juego. 2. Ahorro de recursos: La interfaz gráfica y sus elementos gráficos es uno de los componentes que consume mayor cantidad de recursos, por ende se desea limitar el uso de estos. 3. Control: Al tener solo una instancia de estas clases se facilita el control sobre las mismas para realizar modificaciones. 4. Evitar duplicación: Respecto a temas de configuración del juego, es necesario solo una opción de configuración para no caer en inconsistencias. Finalmente se hace uso del patrón de diseño Singleton ya que dentro del juego solo es necesario instanciar una y solo una vez cada clase para hacer un correcto uso del juego.
Observer	Casilla y Observer	En este caso el sujeto de estudio es la casilla, con sus diferentes estados los cuales están siendo observados por el objeto Observador, el cual desencadena eventos. Los motivos de su uso son: 1. Mantenimiento: Al no haber acoplamiento entre los objetos casillas y observador se facilita el mantenimiento y extensibilidad del código, ya que al momento de realizar cambios en un objeto no se ve afectado el otro. 2. Escalabilidad: Al momento de querer agregar código directamente relacionado con los estados de las casillas no es necesario cambiar el objeto casilla, sino que simplemente se le anexa un observador, haciendo esta tarea más sencilla y permitiendo al programa ser más escalable.

		3. Legibilidad: El código se vuelve más comprensible para nuevos programadores que quisieran agregar funcionalidades al juego, ya que este posee una lógica definida basada en eventos

Otras buenas prácticas utilizadas para mejorar la calidad del software

1.Uso de normas y convenciones de nombres

Convención	Ejemplo	Cumple
PascalCase para clases	InterfazGrafica	Si
camelCase en métodos y funciones	iniciarJuego()	Si
camelCase en nombres de atributos y variables	matrizTablero	Si
Guion bajo para atributos privados	_instance	Si
Mayusculas y guion bajo para constantes	TIEMPO_JUEGO	No aplica

2.Corrección de código en pares

La corrección de código se llevó a cabo por el estudiante Matias Relauquen al código fabricado por el estudiante Rodrigo Vargas. Se llegó a las siguientes conclusiones:

Principios de la Programación Orientada a Objetos (POO)

- Encapsulamiento: Los objetos están encapsulados en clases, y los métodos y atributos están definidos de manera coherente en cada clase.
- Herencia: No se observa un uso extensivo de la herencia en el código, puede no ser necesario en este caso, ya que el juego es simple.
- Polimorfismo: El polimorfismo no se ve explícitamente en el código, puede no ser necesario en este caso

Refactorizaciones y Legibilidad

- Refactorización: Se puede cambiar el código para mejorar la legibilidad, como la utilización de nombres más descriptivos para variables y métodos.
- Comentarios: Se podrían agregar más comentarios para explicar partes específicas del código, especialmente en áreas más complejas o donde la intención no es inmediatamente clara.

Rendimiento

- Rendimiento: Al ser un juego simple no se considera este apartado.

Todos estos comentarios fueron tomados en cuenta para la corrección final del código.

3.Identificación de criterios de calidad

1. Funcionalidad:

- Evaluación: El código funciona correctamente. Sin embargo no es la versión final con todos las funcionalidades requeridas.

2. Usabilidad:

- Evaluación: La interfaz gráfica permite al usuario interactuar y jugar correctamente.

3. Eficiencia:

- Evaluación: El encapsulamiento permite una mayor eficiencia en el código. Sin embargo, al ser un software pequeño ese criterio no es primario.

4. Mantenibilidad:

- Evaluación: Los comentarios y el patrón de diseño observer hacen que este criterio se dé por cumplido.

5. Escalabilidad:

- Evaluación: No es un requisito primario para este juego, pues las acciones a realizar son acotadas.

Conclusión

En resumen, la implementación de la arquitectura de software utilizando la metodología 4+1, la aplicación de patrones de diseño y buenas prácticas ha sido esencial para lograr un producto de software de mayor calidad, ya que estos apartados permiten tener una visión general del código antes de comenzar a construir y corregir errores de manera más sencilla.