Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI0136 Diseño de Software
Prof. Alan Calderon

Justificación del diseño
MARDA: Juego LUDO

Elaborado por:

Javier Nuñez Araya B55078
Alexandra Siles Alvarado B56893
Sebastian Vargas B47431
Juan David Díaz Marchena B72584

Problema

Hacer una implementación del juego Ludo bajo el supuesto que se está trabajando en una empresa donde se desarrollan juegos pertenecientes a la familia de juegos de mesa. A corto plazo se necesitará implementar los juegos: Serpientes y escaleras, Monopoly y Backgammon. Por lo tanto se implementará un MARDA con el fin de reutilizar código a corto, mediano y largo plazo.

Descripción de la solución

Para la implementación de la solución del MARDA se aplicaron varios principios solid, el que más se aplicó fue el Open Closed Principle, porque se aplicó en la generalización del prototipo de LUDO realizado originalmente, donde se modificaron todas las clases que se pudieron modificar para lograr aplicar también el patrón plantilla donde se utilizó como estandar el nombrar las clases tipo: "I_NomClase.py" para identificar cuales son las clases generalizadas utilizadas en patrón plantilla. Además con varios métodos se aplica el patrón de Single Responsibility Principle. También se implementa el patrón MVC para donde se mandela un modelo del Tablero que se encarga de crear la estructura de datos que se utiliza, en nuestro caso es un grafo pero se puede utilizar otra estructura si se implementa otro juego, un controlador que es el que se encarga de las funcionalidades que ocurren en el juego y una parte gráfica que es la encarga de pintar las imágenes en pantalla según los datos que le pase el controlador. También se aplica el patrón plantilla en la parte de Casilla, Casilla Especial, etc. Donde se aplica con Casilla Especial como clase base, y se crean los tres tipos de casillas especiales que son Base, Centro e ingresar Centro que son clases que heredan de Casilla Especial.

Interfaz gráfica: Para la interfaz gráfica se consideró principalmente el uso de las librerías tkinter y pygame. Se decidió usar tkinter debido a que la entrada de datos es

mucho más simple y se ve mejor. Sin embargo durante la elaboración del proyecto se notó que el movimiento de las fichas podría ser más simple con pygame. Se decidió seguir usando tkinter debido a la curva de aprendizaje que representaba la integración de pygame y la necesidad de realizar la entrega del proyecto en una cantidad corta de tiempo. Debido a esto no se restringió el MARDA al uso de alguna librería para la representación gráfica del juego.

Controlador: En el controlador, es donde se verifican y aplican las reglas del juego, es el que se comunica con la interfaz gráfica para que se hagan las modificaciones a las imágenes que se están mostrando o que se van a mostrar y también cumple la función de árbitro del juego, donde nosotros no vimos necesario aplicar una clase aparte que cumpliera ese papel porque las reglas que se implementaron son muy pocas y básicas pero de igual manera, para el MARDA se crearon las clases abstractas para que si en otro juego requieren utilizar un árbitro no tengan problema para implementarlo.

Tablero: Para el tablero se implementó un grafo no dirigido con un diccionario debido a que es la estructura que mejor cumple el principio de Open-Close ya que permite la implementación de cualquier tipo de tablero y no solamente los tablero cuadrados. Además gracias a las características de python y una implementación genérica de las clases Nodos y Casilla se cumple el principio de sustitución de Liskov donde se podrá usar para otros juegos o en el caso del grafo para otra funcionalidad diferente al tablero. Además también cumple el principio de responsabilidad única porque la única función del tablero es servir como estructura contenedora. Adicionalmente se quiere destacar de que si bien el grafo es una estructura de naturaleza “pesada” al ser utilizada para juegos de mesa se aprovecha bien el espacio y se obtiene buen rendimiento gracias a que se implementó con un diccionario. Para los tipos de casillas (etiquetas del grafo) de utilizó el método plantilla para que los diferentes tipos de casillas fueran extensibles.

Árbitro: En nuestro caso, la implementación del árbitro es simbólica porque no tuvimos la necesidad de implementar, porque esa función la cumple el controlador propiamente. Pero de igual manera se agregó en el MARDA para estandarizar con los otros juegos que se están implementando.

Gestor de partidas: Para el gestor de partida se utilizaron solamente métodos plantilla ya que se decidió utilizar la librería pickle que permite la escritura y lectura de clases en binarios que contienen variables primitivas como hilera y enteros. Además esta librería permite trabajar con tipos un poco más complejos pero comunes en Python como listas y diccionarios por lo que mayormente se cumple el principio de sustitución de Liskov. Como ventaja adicional al ser un formato binario no puede abrirse con un editor de texto para modificarse, lo que es una pequeña medida de seguridad.

Inventario: El inventario, igual que el árbitro, no se sintió la necesidad de implementarla porque en el Ludo todo el inventario son 4 fichas entonces parecía que era sobre cargar con una clase inventario, pero igual se implementa en el MARDA porque en otros juegos sí es necesario implementar un inventario. Además en el MARDA se está aprovechando la capacidad de los contenedores de Python de tener diferentes tipos de variables.

Clases del Marda

class I_Controlador(Interface):

Funcion: inicializa los jugadores
Requiere: lista de jugadores
Modifica: Jugadores
Retorna: Jugadores inicializados

iniciarJugadores(self,jugadores):

Funcion: inicializa los jugadores
Requiere: lista de jugadores
Modifica: Jugadores
Retorna: Jugadores inicializados

escogerColores(self):

Funcion: Pone el color de cada jugador
Requiere:
Modifica: Jugador
Retorna: -

moverFicha (self,jugador,ficha):

Funcion: Mueve fichas en el tablero
Requiere: jugador, ficha,tablero
Modifica: Posicion de la ficha
Retorna: -

cambiarTurno(self):

Funcion: Cambia de turno para seguir el orden del juego
Requiere: -
Modifica: jugador en turno
Retorna: -

tirarDado(self):

Funcion: Tira el dado para jugar
Requiere: -

Modifica: -
Retorna: dado

escogerPrimero (self, labT):
Funcion: Escoge el primer jugador de la partida
Requiere: label para informar
Modifica: turnos de jugadores
Retorna: -

cargar(self, archivo):
Funcion: Carga una partida guardada
Requiere: archivo
Modifica: tablero, jugadores, turnoActual y dado
Retorna: -

guardar(self, archivo)
Funcion: Guarda la partida actual
Requiere: partida iniciada
Modifica: archivo
Retorna: -

resetJugadores(self):
Funcion: Reinicia la lista de jugadores
Requiere: --
Modifica: --
Retorna: --

def terminado(self, par1=None, par2 = None, j = None):
Funcion: Determina si un jugador termino o gano la partida
Requiere: --
Modifica:--
Retorna: --

obtenerJugadores(self):
Funcion: Obtiene lista de jugadores
Requiere: --
Modifica: --
Retorna: --

Clase I_Ayuda

mostrarReglas(self)

Función: Muestra las reglas e instrucciones del juego.

Requiere: --

Modifica: --

Retorna: --

Clase I_Inventario

agregar(self,elementoNuevo)

Funcion: Agrega nuevos elementos a la estructura que se define como almacenador

Requiere: El elemento que se desea agregar

Modifica: La estructura de almacenamiento

Retorna: --

quitar(self,elementoEliminar)

Funcion: Elimina elementos existentes en la estructura que se define como almacenador.

Requiere: El elemento que se desea eliminar.

Modifica: La estructura de almacenamiento.

Retorna: --

esta(self,elemento)

Funcion: Busca un elemento en La estructura de almacenamiento.

Requiere: El elemento que se desea buscar.

Modifica: --

Retorna: La posición del elemento si está, sino retorna -1.

Clase I_Grafico

resetMenu(self,jugadores,controlador)

Funcion: Limpia los datos ingresados en la pantalla del menu principal lo que permite ingresar nuevos datos

Requiere: Lista de jugadores no vacia y una instancia de controlador para reiniciar la lista.

Modifica: La lista de jugadores y los aspectos graficos de entrada de datos.

Retorna: --

cargarTablero(self,ventanaMenu,controlador,jugadores)

Funcion: Carga en pantalla la ventana principal con el juego.

Requiere: La ventana del menú desde donde se carga la nueva pantalla y lista de jugadores no vacía.

Modifica: La ventana del menú se oculta.

Retorna: --

resetVenJu(self,windows,ventanaMenu,jugadores,controlador)

Funcion: Elimina la ventana del juego actual y regresa el jugador a la ventana del menú principal

Requiere: La ventana actual del juego que se va eliminar y la ventana del menú principal que cargo la ventana del juego además de lista de jugadores no vacía

Modifica: Ventana actual

Retorna: --

cargar(self,window,controlador,jugadores)

Funcion: Carga una nueva partida previamente guardada

Requiere: Estar en la ventana del juego y tener un archivo valido para cargarlo

Modifica: La partida actual se sobreescribe

Retorna: --

guardar(self,window,controlador)

Funcion: Guarda la partida actual

Requiere: Estar en la ventana del juego.

Modifica: --

Retorna: Genera un archivo con la partida guardada

cambiarTurnos(self,label,controlador,jugadores)

Funcion: Pasa el turno al siguiente jugador

Requiere: Lista de jugadores no vacía y una etiqueta para colocar el color del siguiente jugador

Modifica: La etiqueta

Retorna: --

ventanaMenu(self,jugadores,controlador)

Función: Es la ventana del menú principal desde donde se ingresan los datos del juego

Requiere: --

#Modifica: Añade los jugadores a la lista de jugadores y les asigna el color según los datos ingresados

Retorna: --

ventana(self,jugadores,controlador,venMenu)

Función: Es la ventana principal del juego donde esta el tablero y los demás elementos

Requiere: Lista de jugadores no vacía

Modifica: --

Retorna: --

manual(self)

Función: Muestra el las reglas del juego y las instrucciones generales

Requiere: Una instancia de ayuda

Modifica: --

Retorna: --

Clase I_ModeloTablero

crearTablero(self):

Funcion: Crea el tablero
Requiere: --
Modifica: tablero
Retorna: --

Clase Grafo

__init__(self):

Funcion: Inicializar grafo

Requiere: ---

Modifica:grafo

Retorna: --

agregarNodo(self,nuevoNodo):

Funcion: Agrega un nodo nuevo al grafo

Requiere: nodo nuevo

Modifica: grafo

Retorna: --

obtenerNodo(self,posicion):

Funcion: Devuelve el nodo con la posicion asociada

Requiere: posicion

Modifica: --

Retorna: el nodo con la llave posicio,

agregarArista(self,nodo1,nodo2):

Funcion: Agrega una arista al grafo almacenandola en los nodos

Requiere: nodo1 y nodo2

Modifica: Grafo, nodos

Retorna: --

imprimir(self):

Función: imprime el grafo, este método es solamente para pruebas y por eso imprime en consola

Requiere: tablero inicializado

Modifica: --

Retorna: --

Clase Nodo

__init__(self,etiquetaParam):

Funcion: Constructor del nodo

Requiere: --

Modifica: Nodo

Retorna: --

asignarArista(self,arista, index):

Funcion: Agrega una arista a la lista de aristas del nodo.

Requiere: Arista, indice

Modifica: lista de aristas del nodo

Retorna: --

obtenerPosicion(self):

Funcion: Devuelve la posicion de la casilla.

Requiere: Requiere que el nodo sea de tipo Casilla

Modifica: --

Retorna: posicion de la casilla

obtenerAristas(self):

Funcion: Devuelve las aristas que están relacionadas con el nodo.

Requiere: nodo inicializado

Modifica: --

Retorna: lista de aristas

obtenerEtiqueta(self):

Funcion: Devuelve la etiqueta del nodo.

Requiere: nodo inicializado

Modifica: --

Retorna: etiqueta

obtenerPosicionesRelacionadas(self):

Función: Devuelve una lista con las posiciones de los nodos que estan en la lista de aristas.

Requiere: nodo inicializado

Modifica: --

Retorna: lista de posiciones

modificarEtiqueta(self, nuevaEtiqueta):

Función: Modifica la etiqueta, se usa para cambiar el tipo de casilla después de que se creó el tablero.

Requiere: nodo inicializado

Modifica: etiqueta

Retorna: --

I_Clase Arista

__init__(self,primero,segundo):

Funcion: Constructor de la arista

Requiere: -

Modifica: Arista

Retorna: -

obtenerPrimero(self):

Funcion: Devuelve el nodo 1 de la arista

Requiere: Arista inicializada

Modifica: -

Retorna: nodo

obtenerSegundo(self):

Funcion: Devuelve el nodo 2 de la arista

Requiere: Arista inicializada

Modifica: -

Retorna: nodo

obtenerNodos(self):

Funcion: Devuelve una tupla con los nodos de la arista

Requiere: Arista inicializada

Modifica: -

Retorna: tupla

Clase Casilla

agregarFicha(self,ficha,controlador):
Función: agrega una ficha a la casilla
Requiere: -
Modifica: Casilla
Retorna-

sacarFicha(self):
Función: saca la primera ficha de la casilla
Requiere: Casilla inicializada y no vacía
Modifica: Casilla
Retorna Ficha

obtenerPosicion(self):
Función: Devuelve las coordenadas de la casilla como si fuera una matriz.
Requiere: Que el tablero tenga representación matricial
Modifica: -
Retorna tupla

Clase I_GestorPartidas

cargar(self,archivo)

Función: Lee el archivo binario y devuelve los objetos leidos para retomar una partida.
Requiere: el nombre de un archivo que sea generado con el metodo guardar de este juego
Modifica:No modifica ningun parametro
Retorna: modeloTablero,jugadores,jugadorActual y dado

guardar(self,nombreArchivo,tablero,jugadores,jugadorActual,dado = 0)

Función:Guarda la información necesaria en binario para retomar la partida luego.
Requiere: Tablero,jugador actual y dado se pueda guardar con pikle
Modifica: No modifica ningun parametro
Retorna: No retorna ningun parametro

Clase I_Arbitro

movimientoValido(self,tablero,ficha,posicion=None)

Función: Indica si se puede hacer algún movimiento
Requiere: tablero creado, una posición en el tablero y fichas inicializadas
Modifica: No modifica ningun parametro
Retorna: si el movimiento es válido

ganadorPartida(self,tablero)

Función: Indica el ganador de la partida
Requiere: tablero creado y que contenga fichas
Modifica: No modifica ningun parametro
Retorna: si hay ganado retorna el color

terminarTurno(self,controlador)

Función: termina el turno
Requiere: el juego iniciado
Modifica: Turno actual
Retorna: no retorna ningun parametro

Clase I_Ficha

setPos(self,pos)

Funcion: Asigna la posición de la ficha en el tablero
Requiere: --
Modifica: --
Retorna: --

getPos(self)

Funcion: Devuelve la posición de la ficha en el tablero
Requiere:--
Modifica:--
Retorna: Posición de la ficha.

Clase I_ModeloTablero

crearTablero(self)

Funcion: Crea el tablero
Requiere: --
Modifica: tablero
Retorna: --

clase I_Jugador

getTurno(self):

Funcion: Obtiene el turno del jugador ya sea que es su turno o no
Requiere: El jugador inicializado
Modifica: --
Retorna: retorna el turno del jugador

getColor(self):

Funcion: Obtiene el color del jugador
Requiere:El jugador inicializado
Modifica:--

Retorna: retorna el color del jugador

getNombre(self):

Funcion: Obtiene el nombre del jugador
Requiere:El jugador inicializado
Modifica: --
Retorna: Retorna el nombre

getInventario(self):

Funcion: Obtiene el inventario del jugador
Requiere:El jugador inicializado
Modifica: --
Retorna: Retorna el inventario

setColor(self,color):

Funcion: Pone el color al jugador
Requiere: El jugador inicializado
Modifica: Modifica el color del jugador
Retorna:--

setTurno(self,turno):

Funcion: Pone el turno del jugador ya sea falso o verdadero
Requiere:El jugador inicializado
Modifica: Modifica el turno
Retorna: --

setNombre(self,nombre):

Funcion: Pone el nombre del jugador para identificarlo de los otros
Requiere:El jugador inicializado
Modifica: Modifica el nombre
Retorna: --