

Estructuras de Datos 2022-2

Proyecto 1.

Implementación de un programa para jugar Wizard.

13 de abril de 2022

Alumno: José David García Díaz

Trabajo Individual



Ejecutar el proyecto.

Requiere Java.

Ejecutar el archivo Wizard-1.0-jar-with-dependencies.jar. Para ello podemos abrir una terminal en la carpeta Wizard y escribir

```
java -jar ./target/Wizard-1.0-jar-with-dependencies.jar
```

Compilar y ejecutar el proyecto.

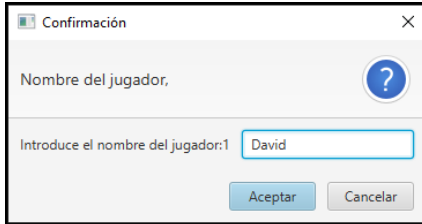
Requiere Maven.

Para ello podemos abrir una terminal en la carpeta Wizard y escribir

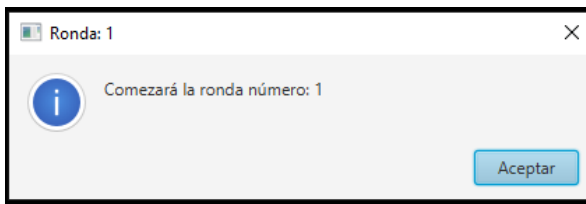
```
mvn install
```

1. Manual de uso.

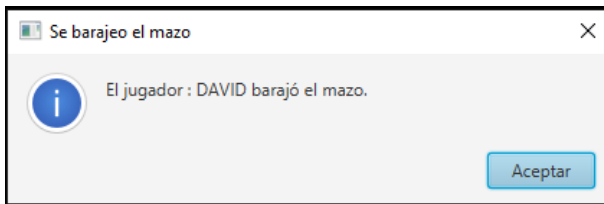
Al iniciar el juego se nos preguntara el número de jugadores y sus nombres, introducimos dichos datos:



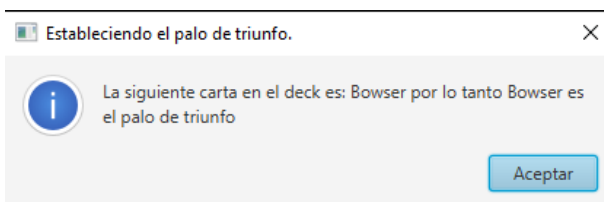
Antes del inicio de cada ronda se nos avisará.



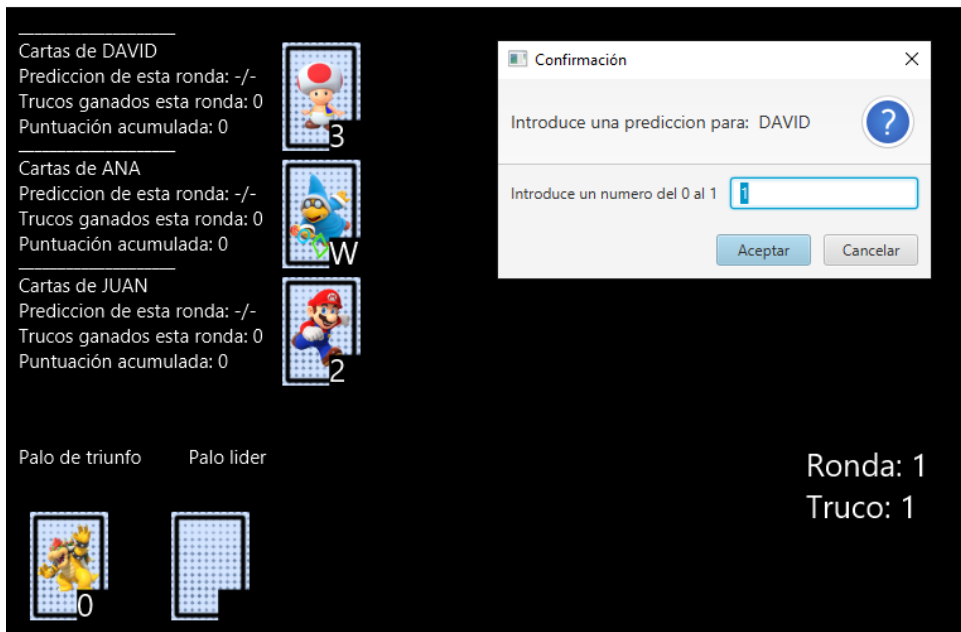
El programa nos dirá quien barajó el mazo.



También nos dirá cuál es el palo de triunfo y la razón:



Ahora veremos que se han repartido las cartas y el programa nos pedirá que ingresemos las predicciones de cada jugador







Una vez indicada la predicción de cada jugador empezará el primer truco y deberemos ingresar la carta que jugará el jugador en turno:



Ingresar una carta

El mazo consta los palos: Peach, Mario, Toad y Bowser a los que nos referiremos por sus iniciales: P, M, T y B respectivamente. Así si queremos ingresar la carta "Toad 3" deberemos escribir "T3".

 Palo: Toad = T	=T3
 Palo: Bowser=B	=B7
 Palo: Mario = M	=M4
 Palo: Peach = P	=P11

Para cartas especiales Shy Guy (actúa como bufón) y Kamek (actúa como Mago) utilizaremos las letras "S" y "W", así:

 Kamek	=W	 Shy Guy	=S
--	----	--	----

Obviamente no podremos jugar cartas que no pertenezcan al jugador en turno o que por las reglas del juego no se puedan jugar.

La cartas que jugó cada jugador aparecerán en la parte inferior, así como el palo de triunfo de la ronda y el palo líder del truco actual.

Ganador del truco y puntuaciones de la ronda.

Una vez ingresadas las cartas de cada jugador se avisará quien ganó el truco:

Wizard by David Diaz

Cartas de DAVID
Predicción de esta ronda: 0
Trucos ganados esta ronda: 0
Puntuación acumulada: 0

Cartas de ANA
Predicción de esta ronda: 1
Trucos ganados esta ronda: 0
Puntuación acumulada: 0

Cartas de JUAN
Predicción de esta ronda: 0
Trucos ganados esta ronda: 0
Puntuación acumulada: 0

Ganador: ANA

 El ganador del truco fue: ANA debido a que jugó un mago primero.

Aceptar

Palo de triunfo

Palo lider

ANA

JUAN

DAVID

0

0

W


2

3

Ronda: 1
Truco: 1

Al terminar la ronda se nos mostrará como cambió la puntuación de cada jugador en esta ronda.

Ronda: 1 finalizada.

 Resultados
DAVID: 0 --> 20
ANA: 0 --> 30
JUAN: 0 --> 20


Aceptar

Historial

En cualquier momento del juego podremos escribir en el dialogo actual la palabra "HISTORIAL" lo que nos mostrará el historial de la partida hasta el momento:

Confirmación

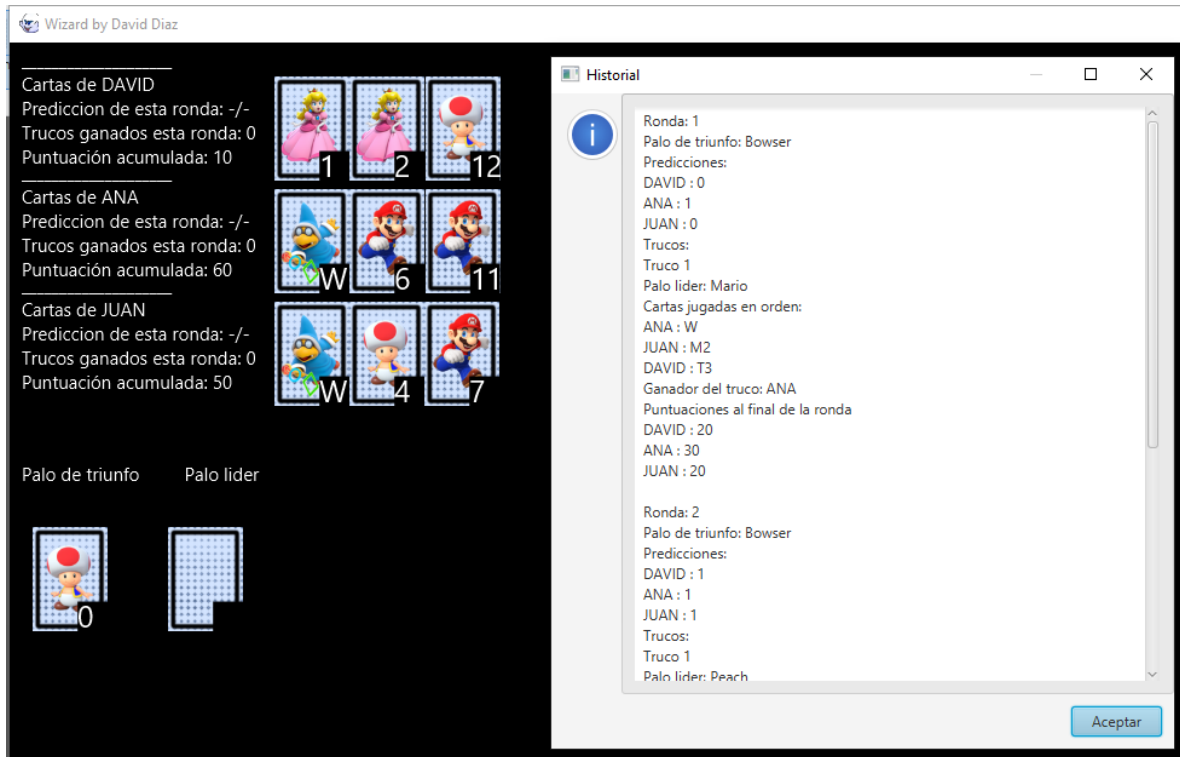
Introduce una prediccion para: DAVID



Introduce un numero del 0 al 3

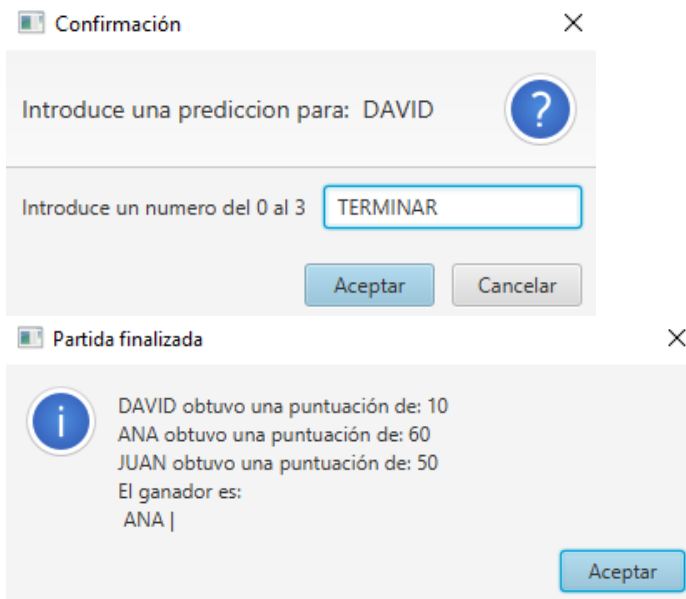
Aceptar

Cancelar



Cerrar la aplicación.

En cualquier momento del juego podremos escribir en el dialogo actual la palabra “TERMINAR” lo que cerrará la aplicación. Se nos mostrarán las puntuaciones hasta el momento y el ganador(o ganadores si hay empates)



2. Estructura general del proyecto.

Procesamiento de trucos y rondas.

Dependiendo del número de jugadores el juego tiene una cantidad determinada de rondas y trucos, básicamente el programa avanza en las rondas y trucos avisándole por medio de alertas al usuario lo que va ocurriendo, cuando el programa requiere input del usuario se presenta una ventana para ingresar el input y el programa continua.

El proyecto consta de una clase llamada GameManager la cual inicia un juego dado el número y los nombres de los jugadores y llama a un ciclo principal:

```
/**
 * Loop which control the whole game
 */
static void mainLoop(){
    int roundNo = 1;
    while(roundNo <= currentGame.getNumberOfRounds()){//Juega el numero de rondas calculado
        StartNextRound(roundNo);//Comienza la ronda
        currentRoundData.setWinnerFigure(currentRound.getWinnerFigure());//Guarda en la data el palo de triunfo
        while(currentRound.getCurrentTrick() <= roundNo){//Ejecuta tantos trucos como rondas
            TrickData data = new TrickData(currentRound.getCurrentTrick());//datos del truco actual para el historial
            currentRound.nextTrick(data);//Ejecuta el siguiente truco
            Player winnerOfTrick = getTrickWinner(currentRound.getWinnerFigure(), currentRound.getLeaderFigure());//Obten el ganador del truco
            data.setWinner(winnerOfTrick);//Establece el ganador en los datos del truco actual
            resetPlayedCards();//Vacía la lista de cartas que están en juego
            App.drawEverything();//Dibuja los datos en pantalla
            currentRound.setLastWinner( winnerOfTrick.getPlayerId() ); //Actualiza el ultimo jugador que gana
            currentRoundData.addTrickData(data);//Agrega los datos del truco actual a los datos de la ronda actual
        }
        String scoreChange = increaseAndSavePlayerScores();
        App.showMessageDialogToUser("Ronda: " + roundNo + " finalizada. ", "Resultados \n" + scoreChange);//
        resetPlayerTrickWins();
        App.drawEverything();
        roundData.add(currentRoundData);//agrega los datos de la ronda actual al historial
        roundNo++;
    }
    App.terminar();
}
```

El desarrollo de cada truco es controlado por `currentRound.nextTrick()`, el cuál le exigirá a cada jugador que juegue una carta.

El método `Player.validateCard(String, int)` validará que la carta introducida por el usuario es válida, es decir si la carta existe, está en la mano del jugador en turno y se puede jugar.

Las cartas que juegue cada jugador así como quien las jugó se guardaran de manera ordenada en la lista de pares `GameManager.cardsPlayed` y una vez que todos los jugadores hayan jugado su carta se establece el ganador del truco con el método `GameManager.getTrickWinner(int, int)` el cual analiza las cartas que se jugaron y determina el jugador que ganó el truco.

Una vez que se jueguen todos los trucos de una ronda, se actualiza la puntuación y se prepara una nueva ronda.

Interfaz.

Para la interfaz se utilizó JavaFx 11, los métodos que dibujan la ventana, envían mensajes al usuario y solicitan datos de entrada al usuario. Cada vez que la información a mostrarse en

pantalla cambia la ventana es redibujada completamente; algo con lo que no estoy muy cómodo pues no es óptimo y generó algo de desorden en el código, quise aprender a usar JavaFx sobre la marcha y debido a esto el uso que le di no es para nada adecuado.

Historial.

Para guardar el historial tenemos las clases RoundData y TrickData para guardar la información relevante de cada ronda y truco respectivamente. Las instancias de TrickData se crean al inicio de cada truco y sus campos se actualizan a medida que progresa el truco. Las instancias de RoundData contienen una lista de objetos tipo TrickData, al finalizar cada truco una instancia de TrickData es añadida a dicha lista en la instancia de RoundData de la ronda actual. RoundData contiene además de otros datos relevantes de una ronda. Así pues la información usada para el historial se encuentra en GameManager.roundData que es una lista de objetos tipo RoundData.

Herramientas utilizadas.

Para estructurar y compilar el proyecto utilicé Maven y como ya mencioné para la interfaz utilicé JavaFx 11.

Estructuras utilizadas.

Para las cartas de los jugadores, las cartas que se estaban jugando y básicamente cada que aparece una lista de cartas utilice una lista simplemente ligada. Sólo en un método (para barajar el mazo) utilicé la pila, sin embargo esto fue absolutamente innecesario, bien pude haber utilizado una lista ligada normal. También debí implementar una clase IteratorLinkedList<T> que implementa la interface Iterator<T> para iterar las listas ligadas que implementé. Por otra parte llegué a requerir asociar a un jugador con la carta que jugó en un truco, para lo cual implemente una clase MyPair<T,E> para guardar pares de objetos.

3. Dificultades.

Entre las mayores dificultades que encontré fue poder estructurar el proyecto usando Maven de manera que funcionara JavaFx, al parecer el hecho de que la main class extienda Application provoca que no se pueda ejecutar la aplicación, para lo que debí crear una main class que no extendiera Application.

Por otra parte tuve problemas para generar un archivo .jar que funcionara, encontré que debía incluir las dependencias dentro del archivo .jar, esto lo logré usando el maven-assembly-plugin en el archivo pom.xml además agregué como dependencias los modulos javafx-graphics de Windows, Mac y Linux.

Poder estructurar mi proyecto de manera que pudiera compilar y ejecutar un archivo .jar que me mostrara una ventana en negro me llevó todo un día de trabajo.

Las siguientes dificultades fueron a la hora de decidir cómo se procesarían las rondas y los trucos así como el input del usuario, utilizar el ciclo mainLoop en GameManager no me pareció la forma más elegante ya que el programa avanza hasta pedir el input del usuario, cuando habría preferido que el programa fuera reactivo al input del usuario.

Una dificultad extra fue aprender a usar JavaFx sobre la marcha, lo peor es que el uso que le dí es muy pobre, sin embargo aprender a hacer las cosas de manera más limpia sé que me habría llevado mucho más tiempo. Este proyecto me llevo más de 30 horas entre aprender, decidir las formas y superar las dificultades anteriores y honestamente me pareció algo difícil.