David Upegui Londoño
Juan Manuel Vera

MANUAL TÉCNICO: Parqués Picapiedras

INTRODUCCIÓN:

Este manual servirá para entender el funcionamiento del juego “Parqués Picapiedras” creado como solución de la Practica I del curso Lógica y Programación III. En este manual encontrará la información relacionada con los métodos e instrumentos usados en el desarrollo de la aplicación.

TECNOLOGÍAS EMPLEADAS:

- JavaScript
- HTML
- CSS
- GitHub

CLASES:

CLASES DE MODELO:
En este apartado están todas las clases que tienen como objetivo abstraer un objeto de la realidad y tienen como métodos los getter y setters de sus respectivos atributos:
- Clase Tab: Representa a las fichas de cada jugador. Tiene los siguiente atributos: (string) color, (string) id, (Square) currentSquare, (string) state y (Boolean) canEat. El atributo currentSquare se refiere a la casilla actual de la ficha y el atributo canEat representa si la ficha se puede comer alguna otra ficha en el turno.

- Clase Player: Representa los jugadores del Parqués. Tiene los siguiente atributos: (array<Tabs>) tabsArray, (string) color, (int) tabsQuantity, (int) order, (boolean) inHouse.

- Clase Square: Esta clase representa a una casilla, extiende de la clase SimpleNode. Tiena atributos (string) color, (string) type, (string) id, (array<Tab>) tabsInside.

- Clase SafeOne: Representa especificamente la primera casilla de seguro que esté en la zona de un color. Extiende de Square y tiene sus mismos atributos.

- Clase SafeTwo: Representa especificamente la segunda casilla de seguro que esté en la zona de un color. Extiende de Square y tiene sus mismos atributos.

- Clase Exit: Representa la casilla de salida. Extiende de Square y tienesus mismos atributos.

- Clase BoardRoad: Representa el recorrido total del tablero. Extiende de la clase CircularNodeList y usa como nodos a las casillas. Contiene como aributos: (array<Square>) houseArray (array<Square>) finishArray. Las casillas del tablero que no sean la casa/cárcel o la casilla final están conectadas en orden formando la lista, mientras, las casillas finales y las casas están guardadas en sus respectivos atributos, además, las casillas de la casa están ligadas a las casillas de salida de cada color

- Clase Turn: Representa a los turnos. Extiende de la clase SimpleNode, tiene los siguientes atributos: (Player) player, (int) timesThrowed, (boolean) doMovement.

- Clase Round: Representa a la ronda. Extiende de la clase CircularNodeList.

- Clase Game: Es la clase que representa al juego. Contiene los siguientes atributos: (array<Player>) players, (Round) round, (BoardRoad) boardRoad, (booelan) isFinished, (array<int>) currentDiceValues, (Turn) currentTurn, (array<string>) colorsArray. La clase colorsArray es un atributo estático

CLASES CONTROLADORAS:

Son clases que se encargan de la lógica de negocio, el manejo visual y el manejo de eventos.

- Clase GameControl: Esta clase contiene los siguientes atributos (Game) currentGame, (HTML element) tabSelected. Ambos atributos son estáticos, de esa manera se pueden acceder a ellos desde cualquier función. En esta clase se encuentran todas las funciones que componen la lógica de negocio.

- Clase UiControl: Esta clase tiene las funciones que se encargan del manejo visual de la aplicación. Refleja la lógica de negocio.

- Clase EventControl: Es la clase que contiene los eventos que son añadidos a algunos elementos de la página

CLASES ADICIONALES:

Son clases que no intervienen directamente en el funcionamiento neto de la aplicación, pero hacen parte importante del desarrollo

- Clase SimpleNode: Es la clase que representa a un nodo simple

- Clase CircularNodeList: Es la clase que representa a una lista simplemente ligada circular. Contiene funciones básicas de una lista ligada que se usaron en el desarrollo

- Clase Timer: Esta clase contiene la función sleep(ms), que detiene el flujo de la función que la llame

- Clase Action: Con esta clase se controlan algunas de las acciones de la aplicación. A continuación, se explicará más a fondo el funcionamiento de la función

FUNCIONES IMPORTANTES:

- GameControl.startGame

Esta función llama a GameControl.orderingRound(), después de ejecutar la función, se ejecuta un ciclo que se detiene solo cuando termina el juego, dentro del ciclo se cambia o se repite turno (depende de si los dados eran pares)

- GameControl.orderingRound

La función se encarga de hacer una ronda, durante la ronda ordena al turno de acuerdo al valor de los dados, finalmente se actualiza el valor de la ronda dentro del juego

- GameControl.PlayTurn
En este turno se define si se juega un turno para salir de la casa o un turno normal.

- GameControl.getOutHouse
Es el turno para salir de la casa, esta función se activa cuando todas las fichas están en casa. Se tiran el dado tres veces o hasta que salga par, si sale par, termina saca las fichas de la casa y termina el turno, si no, solo termina el turno

- GameControl.normalTurn
En esta función se lleva a cabo un turno normal. Primero se evalúa si se han sacado 3 pares seguidos (si ha repetido el turno 2 veces y además, en el turno actual saca par) de ser así, se activa la secuencia para seleccionar una ficha y ganar con ella automáticamente. Si no sucede lo anterior se juega el turno normal, en caso de no haber movimientos posibles, se pasa el turno, en cambio, si hay movimientos posibles, se activan las fichas. Al clickear las fichas se mostrará las casillas posibles a las que se puede mover, si el jugador pulsa la casilla la ficha se mueve y termina el turno. En medio del proceso de esta función, se tiene en cuenta comerse a las fichas, sacar fichas de la casa con números pares, ganar con la ficha, etc.

- GameControl.repeatTurn
Actualiza los valores para repetir turno.

- GameControl.nextTurn
Actualiza los valores para iniciar el siguiente turno.

- GameControl.availableSquares
Retorna las casillas a las que se puede mover una ficha dado un valor en los dados. El arreglo de casillas depende, claramente, de la casilla actual de la ficha.

- GameControl.moveTab
Esta función ejecuta el movimiento de una ficha, actualiza los valores de la ficha como currentTurn, además de que ejecuta la función UiControl.moveTab la cual mueve las fichas en la pantalla.

- Action.generateNewAction
Las acciones creadas en el juego se almacenan en un estático de la clase Action. Esto se hace con el objetivo de ahorrar memoria, de esa manera cada que se genere una nueva acción por medio de la función, esta buscará en el arreglo si hay acciones que ya estén finalizadas y las renombra, en el caso en que no haya funciones finalizadas, entonces crea una nueva acción y la guarda en el arreglo.

- GameControl.gameFinished
Evalúa si el juego está terminado. El juego está terminado cuando cada ficha del jugador que se evalúe tenga el atributo state = ‘finish’. Esta acción se ejecuta únicamente cuando una ficha se mueva a la casilla final.

- GameControl.selectWinnerTab
La función se ejecuta cuando se hayan tirados 3 pares consecutivos, se encarga de mover la ficha clickeada a la casilla final y actualizar los atributos necesarios.

La gran mayoría de estas funcionan logran su cometido gracias a la clase Action, a continuación, el funcionamiento de la clase Action

EL PROBLEMA:

Debido a la naturaleza asíncrona de JavaScript, no se puede pausar todo el procesamiento con la función Timer.sleep, esta solo tiene alcance local en la función que la llame, por lo que, si tengo una función f1 que implemente el sleep y una función f2, sucedería lo siguiente:

```
f1();
f2()
```

- Se ejecuta f1
- f1 ejecuta sleep
- **Se ejecuta f2 (sin terminar f1)**
- Termina de ejecutarse f2.
- Termina de ejecutarse f1

El flujo esperado sería que terminara f1 y luego f2. Aquí entra la clase Action que tiene como principales funciones doAction0 y doAction1.

El control de flujo se basa en revisar el cumplimiento de las condiciones, dentro del ciclo estará una función sleep para no crear un ciclo lo suficientemente pesado como para tumbar la aplicación. El ciclo seguirá hasta que se cumpla la condición, por ejemplo, para esperar que se tiren los dados se procedería de la siguiente manera: se activa el botón de lanzamiento de dados, se entra en el ciclo que tiene como condición “si se tiran los dados, salga del ciclo”, de esa manera el programa estará dentro del ciclo hasta que se lancen los dados.

Con las funciones como GameControl.normalTurn sucede lo mismo. Se crea un ciclo en la función que lo llame y con ese ciclo se verifica si la función a terminado. Para hacer esto se crea una Acción con generateAction(), luego por medio de la acción se ejecuta una función, usando el método doAction(action, función que se quiere ejecutar), action es el mismo objeto acción que llama el método. En el caso en que la función que se quiera ejecutar lleve parámetros, estos se ponen al inicio, así: doAction1(parámetro, action, función que se quiere ejecutar).

Ahora, la función que se quiera ejecutar debe llevar, además, el parámetro action, para así, al terminar la función, se ejecuta actionFinished(), de esa manera se sabrá que la función ha terminado.
Ahora que tenemos la condición de finalización, solo queda hacer un ciclo que se ejecute mientras la función NO se haya terminado

Todo este proceso es un poco complejo pero se entiende más fácil con una revisión al código.