```
import React, {ReactNode, useMemo} from 'react';
import {useWindowDimensions, StyleSheet} from 'react-native';
import {Gesture, GestureDetector} from 'react-native-gesture-handler';
import Animated, {
  useAnimatedStyle,
  useSharedValue,
  withSpring,
} from 'react-native-reanimated';
import {useSafeAreaInsets} from 'react-native-safe-area-context';

interface IPropsDraggableView {
  size: number;
  children: ReactNode;
  onDragEnd: (positionX: number, positionY: number) => void;
}

const SPACE_BTW_CORNER = 15;
const DraggableView: React.FC<IPropsDraggableView> = ({
  children,
  onDragEnd,
  size = 100,
}) => {
  const {bottom: safeBottom, top: safeTop} = useSafeAreaInsets();
  const {width: dWidth, height: dHeight} = useWindowDimensions();
  const {init_x, init_y} = useMemo(() => {
    let init_x = dWidth - size - SPACE_BTW_CORNER;
    let init_y = dHeight - (size + safeBottom + SPACE_BTW_CORNER);
    return {init_x, init_y};
  }, []);
  const x = useSharedValue(init_x);
  const y = useSharedValue(init_y);
  const context = useSharedValue({x: x.value, y: y.value});

  const panGesture = Gesture.Pan()
    .onStart(() => {
      context.value = {x: x.value, y: y.value};
    })
    .onUpdate(event => {
      x.value = event.translationX + context.value.x;
      y.value = event.translationY + context.value.y;
    })
    .onEnd(() => {
      /**
       * on drag end, drag it to the nearest corner of the screen & gives callback on onDragEnd function in props of its
       */
      let positionX = x.value;
      let positionY = y.value;

      if (x.value > dWidth / 2) {
        positionX = init_x;
      } else {
        positionX = SPACE_BTW_CORNER;
      }
      if (y.value > dHeight / 2) {
        positionY = init_y;
      } else {
        positionY = safeTop ? safeTop - 5 : 0 + SPACE_BTW_CORNER;
      }

      x.value = withSpring(positionX);
      y.value = withSpring(positionY);

      if (typeof onDragEnd === 'function') {
        onDragEnd(positionX, positionY);
      }
    });

  const panStyle = useAnimatedStyle(() => {
    return {
      transform: [{translateX: x.value}, {translateY: y.value}],
    };
  }, [x.value, y.value]);

  return (
    <GestureDetector gesture={panGesture}>
      <Animated.View style={[styles.draggableView, panStyle]}>
        {children}
      </Animated.View>
    </GestureDetector>
  );
};

const styles = StyleSheet.create({
  draggableView: {position: 'absolute'},
});

export default DraggableView;
```