```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

// Author: Sam Redfern, sam@psychicsoftware.com
// Tribal City Interactive / Psychic Software

public class PsychicCardsGame : MonoBehaviour {
    // inspector
    [SerializeField] private PsychicCard templateCard;
    [SerializeField] private Sprite[] cardSymbols;
    [SerializeField] private Transform cardPositionMarkers;
    [SerializeField] private float initialDealDelay, perCardDealDelay;
    [SerializeField] private TextMeshProUGUI txtLives;

    // internal
    private static float headbandDataCountdown = 1f;
    private static Vector2 packPosition;
    private static List<List<Vector2>> cardLayouts; // outer list is per game layout, inner l
ist is card positions
    private static int currentGameLayout = 0, currentGameLastLayout = 2; // in each the level
s of the game, we play 3 layouts e.g. 0,1,2
    private static int numActiveCards = 0;
    private static int livesLeft = 3;
    private static List<PsychicCard> gameCards = new List<PsychicCard>();
    private static Vector2 tableCentre; // average position of all cards on table

    // for external access
    public static PsychicCardsGame instance;

    private void Start() {
        instance = this;
        templateCard.gameObject.SetActive(false);

        packPosition = cardPositionMarkers.Find("PackPosition").GetComponent<RectTransform>()
.anchoredPosition;
        cardLayouts = new List<List<Vector2>>();

        int l = 0;
        Transform layout = cardPositionMarkers.Find("Layout"+l);
        while (layout!=null) {
            cardLayouts.Add(new List<Vector2>());
            foreach (Transform t in layout) {
                cardLayouts[l].Add(t.GetComponent<RectTransform>().anchoredPosition);
            }
            //Debug.Log("Layout "+l+" has "+cardLayouts[l].Count+" cards");

            l++;
            layout = cardPositionMarkers.Find("Layout"+l);
        }

        cardPositionMarkers.gameObject.SetActive(false);
    }

    private void Update() {
        if (GameManager.GetGameState()==GameState.PlayingPsychicCards) {
            headbandDataCountdown -= Time.deltaTime;
            if (headbandDataCountdown<=0f) {
                headbandDataCountdown = 1f; // headband data frequency is 1 per sec
                GameManager.WriteLogLine();
            }
        }
    }

    public static void StartGame(int level) { // level is 1-3
        ClearCards();
        PsychicCard.StaticInitNewGame();
        GameManager.SetGameState(GameState.PlayingPsychicCards, level);
        currentGameLayout = (level-1)*3; // there's 3 layouts to complete per game level
        currentGameLastLayout = currentGameLayout+2;
        StartNextLayoutInGame();
        headbandDataCountdown = 1f;
    }

    private static void StartNextLayoutInGame() {
        instance.StartCoroutine( DealGameCards() );
        livesLeft = 3;
        instance.txtLives.text = "- "+livesLeft+" -";
    }

    private static void ClearCards() {
        instance.StopAllCoroutines();
        // hide them all
        for (int i=0; i<gameCards.Count; i++) {
            gameCards[i].gameObject.SetActive(false);
        }
        numActiveCards = 0;
    }

    public static void WinCard(PsychicCard card) {
        if (card.gameObject.activeInHierarchy) {
            card.gameObject.SetActive(false);
            numActiveCards--;
            if (numActiveCards<=0) {
                // game won, so do next layout or back to menu if finished all layouts for ou
r game level
                if (currentGameLayout<currentGameLastLayout) {
                    currentGameLayout++;
                    StartNextLayoutInGame();
                }
                else
                    GameManager.SetGameState(GameState.Menu);
            }
        }
    }

    public static void LostLife() {
        livesLeft--;
        instance.txtLives.text = "- "+livesLeft+" -";
        if (livesLeft<=0) {
            // game lost
            instance.StartCoroutine( GameManager.SetGameStateAfterDelay(GameState.Menu,1f) );
        }
    }

    private static PsychicCard GetNewCard() {
        PsychicCard card;
        // take an unused one if available
        for (int i=0; i<gameCards.Count; i++) {
            card = gameCards[i];
            if (!card.gameObject.activeInHierarchy) {
                card.gameObject.SetActive(true);
                return card;
            }
        }
        // or else instantiate a new one
        GameObject go = Instantiate(instance.templateCard.gameObject);
        go.SetActive(true);
        go.transform.SetParent(instance.templateCard.transform.parent, false);
        card = go.GetComponent<PsychicCard>();
        gameCards.Add(card);
        return card;
    }

    private static IEnumerator DealGameCards() {
        yield return new WaitForSeconds(instance.initialDealDelay);
        WaitForSeconds delay = new WaitForSeconds(instance.perCardDealDelay);

        if (cardLayouts.Count>currentGameLayout) {
            List<Vector2> layout = cardLayouts[currentGameLayout];

            List<int> deck = new List<int>();
            for (int i=0; i<layout.Count/2; i++) {
                // add each symbol twice to our "deck" to select randomly from
                int symbol;
                do {
                    symbol = Random.Range(0, instance.cardSymbols.Length);
                }
                while (deck.Contains(symbol));

                deck.Add(symbol);
                deck.Add(symbol);
            }

            tableCentre = Vector2.zero;

            for (int i=0; i<layout.Count; i++) {
                PsychicCard card = GetNewCard();
                numActiveCards++;
                int symbol = deck[Random.Range(0,deck.Count)];
                deck.Remove(symbol);
                card.DealToTable(packPosition, layout[i], symbol, instance.cardSymbols[symbol
]);
                tableCentre += layout[i];
                yield return delay;
            }

            tableCentre /= layout.Count;

            // tell the cards where the table centre is
            for (int i=0; i<gameCards.Count; i++) {
                if (gameCards[i].gameObject.activeInHierarchy)
                    gameCards[i].SetTableCentre(tableCentre);
            }
        }
        else
            Debug.LogError("PsychicCards does not have layout number "+currentGameLayout);
    }

    public void OnClickedPauseButton() {
        GameManager.PauseGame();
    }

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

// Author: Sam Redfern, sam@psychicsoftware.com
// Tribal City Interactive / Psychic Software

public class PsychicCard : MonoBehaviour {
    private enum CardState {
        MovingToInitialPosition,
        BackFaceUp,
        FrontFaceUp,
        FlippingToFrontFaceUp,
        FlippingToBackFaceUp,
        FadingToRemove
    }

    // inspector
    [SerializeField] private float moveSpeed; // pixels per second
    [SerializeField] private Sprite[] flipFrames;
    [SerializeField] private float timePerFlipFrame; // secs
    [SerializeField] private bool allowManualUnflip = false;

    // internal
    private Image cardImage = null;
    private int cardSymbolIdx;
    private Image cardSymbolImage = null;
    private RectTransform rectTransform = null;
    private Vector2 moveToPos;
    private CardState state;
    private int currFlipFrame;
    private float distFromTableCentre = 50000f; // affects how quickly the card's symbol disp
lays when the player focuses
    private float flipFrameCountdown, headbandDataCountdown = 0f;

    // shared by all cards
    private static WaitForSeconds incorrectCardShowTime = new WaitForSeconds(1f);
    private static float nearestDistFromTableCentre = 2000f; // what distance is the closest
card?
    // as soon as a card starts to flip up, it's added to this
    private static List<PsychicCard> faceUpCards = new List<PsychicCard>();

    public static void StaticInitNewGame() {
        nearestDistFromTableCentre = 2000f;
        faceUpCards.Clear();
    }

    public void DealToTable(Vector2 startPos, Vector2 endPos, int cardSymbolIdx, Sprite cardS
ymbol) {
        if (rectTransform==null) {
            rectTransform = GetComponent<RectTransform>();
            cardImage = GetComponent<Image>();
            cardSymbolImage = transform.GetChild(0).GetComponent<Image>();
        }

        this.cardSymbolIdx = cardSymbolIdx;
        cardSymbolImage.sprite = cardSymbol;
        cardImage.color = Color.white;

        state = CardState.MovingToInitialPosition;
        rectTransform.anchoredPosition = startPos;
        moveToPos = endPos;

        SetFlipFrame(0);
        SetSymbolOpacity(0f,false);
        headbandDataCountdown = 1f;
        distFromTableCentre = 50000f; // so we won't see focus effect until all cards are dea
lt and table centre is known
    }

    private void SetFlipFrame(int frameIdx) {
        currFlipFrame = frameIdx;
        cardImage.sprite = flipFrames[frameIdx];
        rectTransform.sizeDelta = cardImage.sprite.textureRect.size;
    }

    private void Update() {
        if (state==CardState.MovingToInitialPosition) {
            Vector2 pos = rectTransform.anchoredPosition;
            Vector2 diff = moveToPos - pos;
            float targDist = diff.magnitude;
            float moveDist = moveSpeed * Time.deltaTime;
            if (targDist<=moveDist) {
                pos = moveToPos;
                state = CardState.BackFaceUp; // arrived at initial position
                headbandDataCountdown = 0f;
            }
            else {
                pos += diff.normalized * moveDist;
            }
            rectTransform.anchoredPosition = pos;
        }
        else if (state==CardState.FlippingToFrontFaceUp) {
            flipFrameCountdown -= Time.deltaTime;
            if (flipFrameCountdown<=0f) {
                flipFrameCountdown += timePerFlipFrame;
                SetFlipFrame(currFlipFrame+1);
                if (currFlipFrame+1>=flipFrames.Length) {
                    // flipping to front face is complete
                    state = CardState.FrontFaceUp;
                    SetSymbolOpacity(1f,false);
                    int idx = faceUpCards.IndexOf(this);
                    if (idx%2==1) {
                        // this was the second in a pair that has been flipped over
                        PsychicCard otherCard = faceUpCards[idx-1];
                        faceUpCards.Remove(this);
                        faceUpCards.Remove(otherCard);

                        if (cardSymbolIdx==otherCard.cardSymbolIdx) {
                            // match
                            FadeAndRemove();
                            otherCard.FadeAndRemove();

                        }
                        else {
                            // no match
                           StartCoroutine( FlipToBackFace(true) );
                            otherCard.StartCoroutine( otherCard.FlipToBackFace(true) );
                            // lose a life
                            PsychicCardsGame.LostLife();
                        }
                    }
                }
            }
        }
        else if (state==CardState.FlippingToBackFaceUp) {
            flipFrameCountdown -= Time.deltaTime;
            if (flipFrameCountdown<=0f) {
                flipFrameCountdown += timePerFlipFrame;
                SetFlipFrame(currFlipFrame-1);
                if (currFlipFrame==0) {
                    // flipping to back face is complete
                    state = CardState.BackFaceUp;
                    headbandDataCountdown = 0f;
                }
            }
        }
        else if (state==CardState.FadingToRemove) {
            Color c = cardImage.color;
            c.a -= Time.deltaTime;
            if (c.a<=0f) {
                PsychicCardsGame.WinCard(this);
            }
            else {
                cardImage.color = c;
                cardSymbolImage.color = c;
            }
        }

        headbandDataCountdown -= Time.deltaTime;
        if (headbandDataCountdown<=0f) {
            headbandDataCountdown = 1f; // headband data frequency is 1 per sec
            if (state==CardState.BackFaceUp) {
                SetSymbolOpacity((MonitorData.attention-40f)/50f, true);
            }
        }
    }

    public void OnClicked() {
        if (state==CardState.BackFaceUp && faceUpCards.Count<4) {
            FlipToFrontFace();
        }
        else if (state==CardState.FrontFaceUp && allowManualUnflip) {
            StartCoroutine( FlipToBackFace(false) );
        }
    }

    private void FlipToFrontFace() {
       state = CardState.FlippingToFrontFaceUp;
        faceUpCards.Add(this);
        flipFrameCountdown = timePerFlipFrame;
        SetSymbolOpacity(0f,false);
    }

    private IEnumerator FlipToBackFace(bool afterDelay) {
        if (afterDelay)
            yield return incorrectCardShowTime;

       state = CardState.FlippingToBackFaceUp;
        flipFrameCountdown = timePerFlipFrame;
        SetSymbolOpacity(0f,false);
    }

    private void FadeAndRemove() { // this is done after a successful match
        state = CardState.FadingToRemove;
    }

    private void SetSymbolOpacity(float opacity, bool reduceWithDistFromTableCentre) {
        if (reduceWithDistFromTableCentre)
            opacity *= nearestDistFromTableCentre/distFromTableCentre;

        opacity = Mathf.Clamp(opacity,0f,1f);

        cardSymbolImage.color = new Color(1,1,1,opacity);
    }

    public void SetTableCentre(Vector2 cPos) {
        distFromTableCentre = (moveToPos-cPos).magnitude;
        if (distFromTableCentre<nearestDistFromTableCentre)
            nearestDistFromTableCentre = distFromTableCentre;

        //Debug.Log("dist="+distFromTableCentre+", nearest="+nearestDistFromTableCentre);
    }
}
```