Слагалица

Овој проект има цел созадавње на игра слагалица во Unity системот. Идеата на апликацијата е да се емулира дигитално редење на слагалици. Играта содржи засега четири слики кои по случаен избор се селектираат за корисникот да ги состави.





Упатство за користење

Кога ќе се вчита апликацијата ја прикажува избраната слика. Со притискање на копчето „Start Game“ делчињата од слагалицата се мешаат. По завршување со мешање на играчот му се овозможува да ги реди деловите.

Кога ќе се наредат сите делчиња на соодветните места играта завршува и играчот може да избере дали сака да игра уште еднаш. Ако го притисне копчето „Start Game“ ќе се избрише завршената слика и ќе се прикаже нова.

Исто така играчот може во секој момент да ја исклучи апликацијата со притискање на „X“ копчето во горниот десен агол на екранот.

Правила

Играта има единствено правило, сите деловите од слагалицата треба да се стават на нивните соодветни места за да се дојде до победа.

Претставување на проблемот

Сечење на сликата

За сликата да се исече на повеќе делчиња кои се совпаѓаа едно со друго искористив повеќе класи кои ja имплементираат Безјеровата крива. Скриптите за класите BezierCurve.cs и BezCurveTemplate.cs, го содржат кодот каде се имплементираат математичките функции за цртање на самите криви. Во BezierCurve.cs се сместени методите за креирање на точките во кривата и самата крива во дво-дименсионални и тро-дименсионални вектори. BezCurveTemplate.cs се корисити како темплејт за тоа како ќе изгледаат исечените страни на делчето од слагалицата.

Во класата Tile.cs се чуваат информации за креирање на едно делче од слагалицата. Поточно, содржи енумерации за страната на делот, енумерација за типот на крива која ќе ја има дадената страна, големина на делот, мапа за кој тип на кривина соодветствува на секоја страна, листа од сите кривини на делченто и информации за пикселите кои се содржат во делот.

public enum Directions { Up, Down, Left, Right }

public enum CurveType { Positive, Negative, Edge }

public static int tileSize = 100;

private Dictionary<(Directions, CurveType), LineRenderer> lineRender = new Dictionary<(Directions, CurveType), LineRenderer>();

public static List<Vector2> bezCurve = BezierCurve.PointList2(BezCurveTemplate.templateControlPoints, 0.001f);

Методите TranslatePts , InvertYCoordinate и SwapAxis служат за трансформирање на у-оската на кривата во зависностс од тоа каков тип на искривеност ќе ја има дадената страна на исечокот. Тие се повикуваат во методот MakeCurve кој со помош на два switch case функционалности вгнездени еден во друг ги разгледува сите можни комбинации на насока и тип на кривина. За пример да го земеме горниот крај на делчето. Ако има позитивна кривина тоа значи дека од горниот дел се испакнува делче и треба да се повика метод за транслација да правилно да се позиционира кривата; ако има негативна крива делчето има вдлабнатина па затоа трба да се преврти кривата и потоа со транслација коректно да се позиционира; и ако горниот дел е раб на слагалицата треба да се отстрани било каква крива од таа стрната. За долната, десната и левата страна примената е слична со мали измени во повикот на функциите.

Методите HideAllCurves и DestroyAllCurves ги кријат и бришат соодветно делчињата со исечени криви. На крај го имаме методот Draw кој прима аргументи за страната и типон на кривата. Овој метод е одговорен за исцртување на кривите линии врз делчето. Прво методот проверува дали линијата е предходно дефинирана и ако не е ја додава во мапата од лини. Потоа се дефинира бојата на линијата и се именува новиот објект. На крај се превземат сите точки од кривата и се предаваат нивните позиции на lineRenderer. Со тоа имаме созададено маска за едно делче од слагалицата.

Маската на секое делче откако ќе се исцрта, треба да се пополни со точните пиксели кои се наоѓаат на тој дел од сликата, па затоа потребно е да се чуваат информации за пикселите.

public Stack<Vector2Int> visitedPixels\_stack = new Stack<Vector2Int>();

public int pixelX = 0;

public int pixelY = 0;

За пополнување на новиот дел користам алгоритам наречен FloodFill кој ги поминува сите пиксели од тој дел на сликата и ги споредува со позицијата на пикселот на даденото делче. Пикселите одкако ќе се обележат дека се посетени се сместуваат во магацин (стек). Кога ќе ги излиста сите пиксели, почнува да се движи низ магацинот (стекот) од пиксели и делчето се обојува со читање и бришење на пиксели од магацинот (стекот). По обојување на еден пиксел се проверуваат горниот, долниот, левиот и десниот соседен пиксел дали се обоени и ако не се се враќаат во магацинот за да бидат повторно процесирани. Методата Fill се повикува за секој пиксел и во неа се доделува бојата на истиот.

void Fill(int x, int y)

{

Color c = ogtexture.GetPixel(x + pixelX \* tileSize, y + pixelY \* tileSize);

c.a = 1.0f;

finaltexture.SetPixel(x, y, c);

}

Штом ќе се изминат сите пиксели се додава новата слика на променливата „public Texture2D finaltexture { get; private set; }“ која враќа информации за делчето до главната логика за исечената слика.

Штом ќе се исече сликата треба сите делчиња да се измешаат и да можат истите да се движат. За таа цел служи класата Movement. Таа има методи за кликање на глувчето врз објек, отпуштање на глувчето врз еден објект и влечење на објектот со глувчето. Кога делчето од слагалицата е поставено на неговата точна позиција, овие методи се враќаат пред да почне извршувањето на логиката на методот.

private void OnMouseUp()

{

if (!GameApp.Instance.MovementEnabled) { return; }

if (EventSystem.current.IsPointerOverGameObject())

{

return;

}

float distance = (transform.position - CorrectPos()).magnitude;

if (distance < 20.0f)

{

transform.position = CorrectPos();

OnTilePlace?.Invoke(this);

}

}

Класата BoardGen ја содржи целата логика за однесувањето на сликата. Тука се сместени методите за генерирање на сликата која ќе се сложува, сечење на сликата, мешање на слагалицата штом ќе се исечат делчињата, водење сметка за тајмерот и декларирање на слагалицата како објект од играта.

Menu класата ги содржи функционалностите на менито. Имено тука се контролира кој екран ќе се покаже, како и што да прави играта при кликање на копчињата за старт на нова игра и ислкучување на апликацијата.

На крај ја имаме скриптата за Singleton претставува шема која ја користи класата GameApp за да се осигура дека за време на играта само една инстанца од објектите на таа класа. Ѕоа значи дека нема да има преклопување на повеќе сцени со слагалици. Класата GameApp складира информации за дали е дозволено движење на некој објект, колку време поминало од почетокот на активирање на објектот, колку делови се на точните позиции и листа од сите можни опции за слики кои се користат во играта за сложувалки.

public bool MovementEnabled { get; set; } = false;

public double SecondsSinceStart { get; set; } = 0;

public int TotalCorrectTiles { get; set; } = 0;

[SerializeField]

List<string> jigsawnames = new List<string>();

Единствениот метод во оваа класа служи за по случаен избор да селектира која слика ќе се сложува.

Со тоа, можеме да ја започнеме играта. Со среќа!