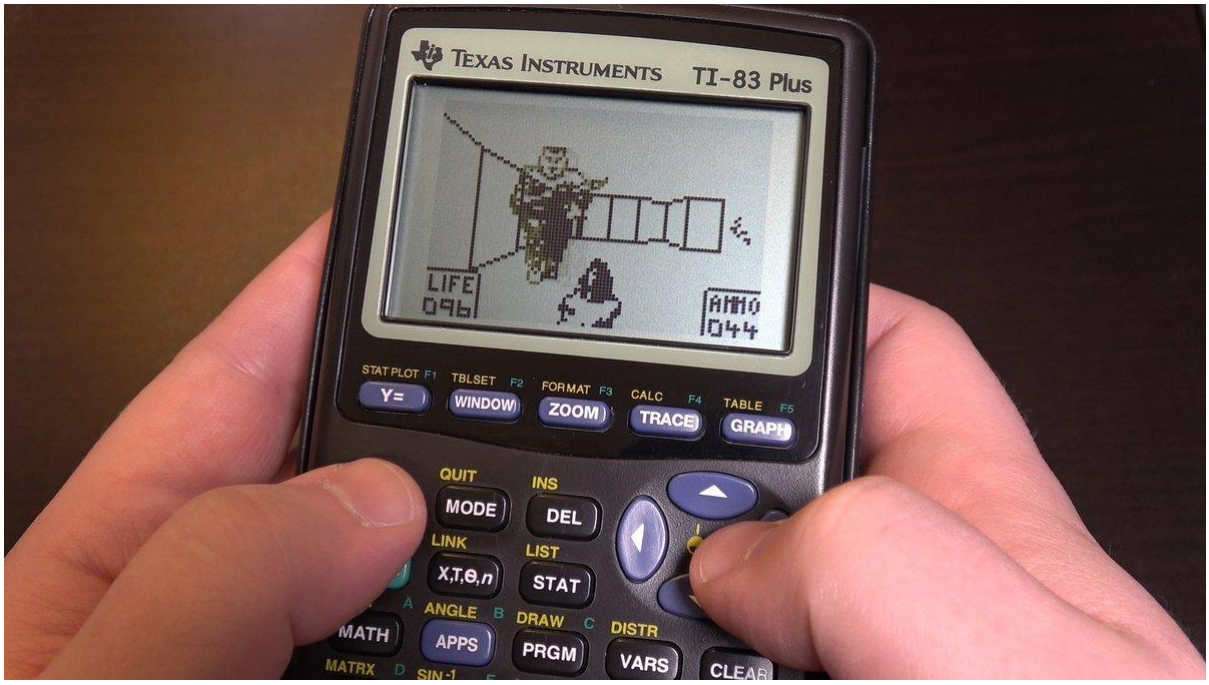
Multiplatform
"Wreck It Richard!"



Sten Duindam
3019434
Multiplatform Development
GDEV2

Inhoudsopgave

Inleiding

Voor MultiPlatform Development heb ik mijn HybridSpace project "Wreck it Richard!" multiplatform gemaakt. Dit omdat het spel goed geschikt is voor meerdere platformen. Zo heb ik uiteindelijk gekozen voor de volgende onderscheidingen:

- Windows
- Arduino
- Iphone
- Android

Ik heb gekozen voor deze onderdelen omdat ikzelf een Iphone heb en het graag zou willen laten zien aan mensen op een eenvoudige manier. Mijn teamleden hebben Android waardoor ik ook hiervoor heb gebuild. Het spel gebruikt in de ideale setting de Arduino als input omdat spelers tijdens het spel op een interactieve muur slaan. Ook heb ik Windows toegepast.

Game Concept

In het kort:

In Wreck it Richard! wordt de speler belaagd door een boze buurman. De buurman wilt in jouw tuin zijn huis uitbreiden en is al begonnen met de voorbereidingen. Dit kan jij natuurlijk niet toelaten en besluit zelf actie te ondernemen! Met de hamer slaat de speler alle dozen van de buurman kapot waarna de buurman alles op alles zet om je te verslaan.

Game flow:

De speler slaat met een schuime hamer op een interactieve muur, klikt met een muis of tapt met zijn vinger op het scherm om de dozen te vernietigen. Dozen kunnen attributen hebben waardoor er een reactie volgt. Wanneer de buurman een rij tot de maximale hoogte heeft gestapeld is de speler af en mag de speler met één laatste slag de buurman toch dwarsbomen.

Code opzet

Gebruikte classes en logica

Omdat het spel vrijwel hetzelfde werkt op ieder platform vond ik het belangrijk om niet te veel afhankelijk te laten zijn van het specifieke platform. Het grootste verschil zit hem in beeldscherm afmetingen en de input. Met dit in gedachte heb ik onderstaande scripts aangepast en ontworpen.

Manager.cs

Het spel bevat een manager class die het spel dirigeert. Vanuit de manager worden o.a. audio, events, user input en gamestates bijgehouden en beïnvloed. Bij ieder platform is deze dus hetzelfde.

De manager vraagt in de awake op welk platform hij draait. Afhankelijk van het platform krijgt hij een AbstractFactory class terug.

```
//Get factory
platform = AbstractFactory.GetInstance();
platform.LoadBoardPreset();
```

Naast de twee onderstaande functies in de manager voor de controls regelt de AbstractFactory alles.

```
if (arduino != null) {
    arduino.setInput(false);
    }
}
```

```
//ARDUINO

public void AddArduinoComponent() {
    this.gameObject.AddComponent<ArduinoRead>();
    }

//END OF ARDUINO
```

De AddArduinoComponent functie is nodig in de manager omdat dit een MonoBehaviour is. Op deze manier kan de AbstractFactory het arduino component toevoegen.

Vervolgens bevat de manager in de Update() een Input controle. Op deze manier maakt het niet uit welke input er gegeven wordt, door de AbstractFactory checkt hij enkel die van het juiste platform en is er geen behoefte om ieder platform los te controleren.

```
private void Update() {
    //Check for input
    platform.CheckInput();
```

AbstractFactory.cs

De AbstractFactory class controleert op welk platform je draait en geeft deze door aan de manager zoals hieronder te zien:

```
public abstract class AbstractFactory {

    public static AbstractFactory GetInstance() {
    SerialPort serialPort = new SerialPort("COM3", 9600);

        //Check if in editor, return windows implementation if this is the case
        if (Application.isEditor)
            return new WindowsImplementation();
        switch (Application.platform) {
#if !DISABLE_SYSTEM
            case RuntimePlatform.WindowsPlayer:
                //Check if there is an arduino, if that is the case run the Arduino implementation
                if (serialPort != null) {
                    return new ArduinoImplementation();
                    }
                //Otherwise continue with the normal WindowsImplementation
                else {
                    return new WindowsImplementation();
                    }
            case RuntimePlatform.IPhonePlayer:
                return new IPhoneImplementation();
            case RuntimePlatform.Android:
                return new AndroidImplementation();
#endif
            default:
                return new DummyImplementation();
        }
    }
    //Two functions to change the game and input per platform
    public abstract void LoadBoardPreset();
    public abstract void CheckInput();
}
```

In de AbstractFactory zijn twee functies. Één om input te controleren en uit te voeren en één om de presets van het spel aan te passen aan het platform(Camera settings en x en y coördinaten waarmee het bord gecreëerd wordt). Op deze manier hoeft de manager verder niet te weten op welk platform hij draait, alles wordt netjes aangegeven door de AbstractFactory.

Hieronder een voorbeeld van de presets.

```
//WINDOWS
public class WindowsImplementation : AbstractFactory {
    public override void LoadBoardPreset() {
        //board presets
        Manager.Instance.xWidth = 5;
        Manager.Instance.yLength = 5;
        Manager.Instance.timeToSpawn = 1;
        //camera setting
        Camera.main.transform.position = new Vector3(2, 3.5f, -5);
        Camera.main.GetComponent<Camera>().orthographic = false;
        }
```

En hier een voorbeeld van input op windows en android(Mouse & Touch)

```
public override void CheckInput() {
    //Input key
    if (Input.GetMouseButton(0)) {
        Vector3 MouseLocation = Input.mousePosition;
        Vector3 sourcePos = new Vector3(Camera.main.transform.position.x, Camera.main.transform.position.y, Camera.main.transform.position.z);
        MouseLocation.z = -sourcePos.z;
        Vector3 targetPos = Camera.main.ScreenToWorldPoint(MouseLocation);

        //Change gamestate to tutorial if the game has not yet started
        if (Manager.Instance.inStartGame) {
            Manager.Instance.inTutorial = true;
            //Set the blockspawner to tutorial mode
            Manager.Instance.blockSpawner.inTutorial = Manager.Instance.inTutorial;
            Manager.Instance.StartGame();
            return;
            }

        //Determine direction of raycast
        Vector3 direction = targetPos - sourcePos;

        //Make the actual raycast
        Debug.DrawRay(sourcePos, direction, Color.red);
        RaycastHit hit;
        if (Physics.Raycast(sourcePos, direction, out hit, 200f)) {
            if (hit.transform.GetComponent<BaseClass>()) {
                hit.transform.GetComponent<BaseClass>().Hit();
                }
            if (hit.transform.GetComponent<NeighbourScript>()) {
                Debug.Log("HIT!");
                hit.transform.GetComponent<NeighbourScript>().HitTheNeighbour();
                }
            if (hit.transform.tag == ("Cat")) {
                Manager.Instance.audioManager.Sfx(blockType.cat);
                }
            }
        }
    }
```

```
public override void CheckInput() {
    //Input Touch
    if (Input.touchCount > 0) {
        Vector3 touchLocation = Input.GetTouch(0).position;
        Vector3 sourcePos = new Vector3(Camera.main.transform.position.x, Camera.main.transform.position.y, Camera.main.transform.position.z);
        touchLocation.z = -sourcePos.z;
        Vector3 targetPos = Camera.main.ScreenToWorldPoint(touchLocation);

        //Change gamestate to tutorial if the game has not yet started
        if (Manager.Instance.inStartGame) {
            Manager.Instance.inTutorial = true;
            //Set the blockspawner to tutorial mode
            Manager.Instance.blockSpawner.inTutorial = Manager.Instance.inTutorial;
            Manager.Instance.StartGame();
            return;
            }

        //Determine direction of raycast
        Vector3 direction = targetPos - sourcePos;

        //Make the actual raycast
        Debug.DrawRay(sourcePos, direction, Color.red);
        RaycastHit hit;
        if (Physics.Raycast(sourcePos, direction, out hit, 200f)) {
            if (hit.transform.GetComponent<BaseClass>()) {
                hit.transform.GetComponent<BaseClass>().Hit();
                }
            if (hit.transform.GetComponent<NeighbourScript>()) {
                Debug.Log("HIT!");
                hit.transform.GetComponent<NeighbourScript>().HitTheNeighbour();
                }
            if (hit.transform.tag == ("Cat")) {
                Manager.Instance.audioManager.Sfx(blockType.cat);
                }
            }
        }
    }
```

Wanneer de speler het spel opent in de Unity Editor krijgt hij de windows controls. De dummy controls sluiten de applicatie omdat er dan geen input mogelijk is en dus het spel niet te spelen is.

ArduinoRead.cs

In een los script is de arduino te vinden, deze wordt toegevoegd aan de manger als component.

```
//ARDUINO
    public class ArduinoImplementation : AbstractFactory {
        public override void LoadBoardPreset() {
        //Create Arduino Component
        if (Manager.Instance.arduino == null) {
            Manager.Instance.AddArduinoComponent();
            Manager.Instance.arduino = Manager.Instance.gameObject.GetComponent<ArduinoRead>();
            }
        //board presets
        Manager.Instance.xWidth = 5;
        Manager.Instance.yLength = 5;
        Manager.Instance.timeToSpawn = 1;
        //camera setting
        Camera.main.transform.position = new Vector3(2, 3.5f, -5);
        Camera.main.GetComponent<Camera>().orthographic = false;
        }

        public override void CheckInput() {
        //Is handled by ArduinoRead
        }
    }
```

In dit script wordt alles geregeld met betrekking tot het opvragen van arduino input tot het omzetten van de gegeven data vanuit de arduino naar bruikbare data voor het spel. Om deze reden is de CheckInput() van de arduino implementatie ook leeg.

Uitdagingen

De grootste uitdaging was voor mij het concept. Ik was erg lang bezig met het bedenken van een game concept, hierdoor ben ik later van start gegaan. Aller eerst was het mijn plan een spel te maken voor webGL en iphone, vanwege dezelfde reden als hybridspace, om in mijn portfolio te kunnen zetten of om overal te kunnen laten zien. Het plan was een Hit em up arcade game. Echter heb ik een windows laptop en na het proberen te builden voor Iphone kwam ik erachter dat een windows laptop niet voor Iphone kan Build & Run'en. Dit moest dus anders.

Vervolgens kreeg ik tijdens ski vakantie het idee om een mobile game te maken waarbij de speler een skileraar was en met touch controls een weg moest swipen over een piste. De skileraar en leerlingen zouden dan dit pad volgen. Voor dit concept heb ik een lineRenderer teken component en waypoint pathfinding AI geschreven(te vinden als filmpje in de github), alleen verloor ik snel motivatie omdat ik zelf geen android telefoon heb. En tijdens vakantie kon ik niet naar de uitleen.

Na de kerstvakantie kwam de realisatie dat ik ook mijn HybridSpace project multiplatform kon maken. Ik dacht er niet bij na om dat voor dit vak uit te voeren ook al waren de plannen er wel.

Toen ik eenmaal het concept had ben ik langzamerweg de code van het project gaan aanpassen om op hele specifieke plekken o.a. de input en afmetingen aan te roepen. Zodat ik het,wanneer ik het multiplatform zou maken, eenvoudig kon vervangen door één oproep(nu de CheckInput()). Arduino had al een eigen script, dus deze kon ik goed weglaten en pas aanmaken wanneer er daadwerkelijk een was aangesloten. De muis input heb ik vertaald van OnMouseDown naar raycasts. Dit was wel spannend want de deadline van HybridSpace kwam er ook aan, maar gelukkig werkte alles.

En nu heb ik dus een mutliplatform HybridSpace product!