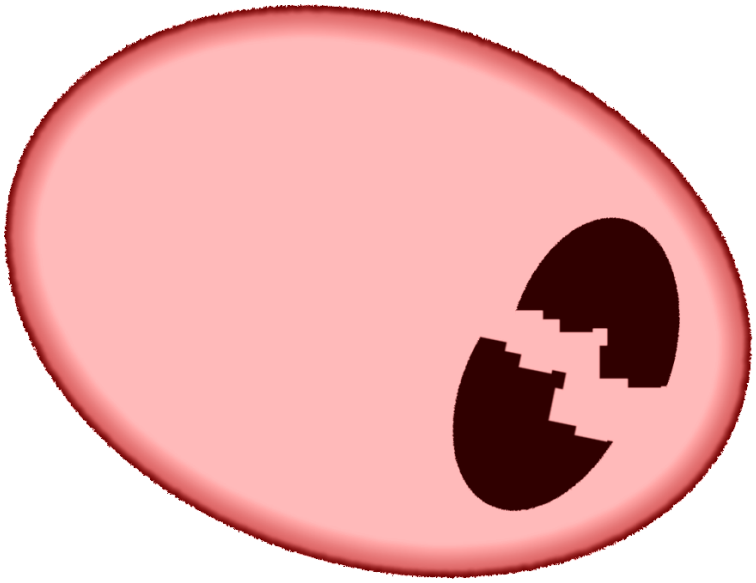
Documentatie - CellInvaders

Jeffrey Saydam - GDV Kernvak 1



Inhoudsopgave

De game 3

Twist 3

Mechanics 3

Design keuzes 3

Plan van aanpak 4

Korte toelichting op Code Ontwerp (Design Patterns) 4

Toepassing van Design Patterns op Code 5

Oude Code (Zonder onderzoek Design patterns) 5

Nieuwe Code (Re-design van code-structuur) 9

Class Diagram Ontwikkeling 12

Class Diagram 1 12

Class Diagram 2 12

Bronnen 13

De game

CellInvaders is een 2D game gemaakt in Unity. CellInvaders is afgeleid van de retro game “Space Invaders”.

Twist

De twist is dat de cellen zich weer kunnen respawnen, totdat men de “kerncel” vernietigd. Hierdoor wordt de game veel moeilijker en uitdagender. Nog een twist is het “vriessysteem”. Door deze mechanic is het niet mogelijk om de Kerncell in één keer te kunnen vernietigen. Zolang een parent-cell een child-cell heeft wordt de parent-cell bevroren en ondoodbaar.

Mechanics

- De speler kan naar links en rechts bewegen met “A en D” of met “Left en Right Arrow”.
- De goep cellen bewegen van link naar rechts.
- De kerncellen spawnen na een tijd subcellen (max 2).
- Het vriessysteem zorgt ervoor dat de parent-cellen niet gedood kunnen worden zolang er subcellen aanwezig zijn in dezelfde celltak.
- De kerncellen spugen om de tijd ziektevagen.
- De speler begint met 3 levens, door ziektevagen worden dezen verminderd.
- De speler heeft een tijd om zo zijn record te verbreken.
- De speler kan zien hoeveel cellen er op het moment aanwezig zijn.
- De speler kan wittecellen schieten om de ziektecellen te vernietigen met spatie.

Design keuzes

- De speler beweegt snel en statisch van links naar rechts om de speler een gevoel van controle te geven.
- De groepcellen bewegen in grote stappen om zo het gevoel van “een retro game” te geven. Ook om het gevoel van de orginele game Space Invaders te geven.
- De twist dat de cellen terug kunnen spawnen maakt het spel extra moeilijk, frusterend. Ook word de speler geactiveerd om techniekten te gebruiken om het spel te winnen.
- Het vriessysteem is om een bepaalde techniek tegen te gaan, dat de speler de kerncel in één keer kan vernietigen.
- De tijd zorgt ervoor dat de speler gemotiveerd blijft om het spel nogmaals te spelen en om zo zijn record te verbeteren.
- De speler kan zoveel wittecellen afschieten als diegene kan drukken met spatie om zo zijn stress op de space button te uiten.
- De art is simplistisch en strak voor een iets moderne gevoel.

Plan van aanpak

Mijn game “CellInvaders” ben ik momenteel in Unity2D aan het maken. Daarin maak ik gebruik van de standaard component (zoals, Collider2D, Rigidbody2D en AudioSource). De rest codeer ik zelf. Ik maak via mijn code veel gebruik van de SetActive() methode om zo Destroy() te illusioneren. Dit is minder efficiënt voor het geheugen, maar wel efficiënt om buggs tegen te gaan. Uit ervaring is Unity een zwakke engine en dit is mijn grootste risico. “slim coderen” zal niet goed werken in Unity, daar ben ik nu met dit project ook al achtergekomen. Ik heb mijn code dus moeten verdommen door alles letterlijk op te schrijven, zodat Unity niks kan overslaan. Als dan alles werk zoals ik dat wil, dan ga ik nog kijken of ik het wat efficienter kan gaan programmeren. Alhoewel dat niet echt nodig zal zijn aangezien dit een kleine 2D game is.

Korte toelichting op Code Ontwerp (Design Patterns)

Structural Patterns
De cellsysteem is de main mechanic van mijn game “Cell Invaders”. Mijn code is gebaseerd uit twee design paterns: Composite en Facade.

De Composite design patern kan men goed terug zien in de cell hierarchy van de game. Elke cell kan onder zich twee nieuwe cellen reproduceren, daardoor ontstaat er een tree-structure. Door deze tree-structure kunnen de parents de children in de gaten houden en kunnen de children de parents op de hoogte blijven houden. Door deze tree-structure die de cellen (de enemy) versterkt wordt de game moeilijker om te halen.

Helemaal aan de top van de tree-structure zit de CellCollection class die men kan zien als de Facade. Deze hele class zorgt ervoor dat al de andere classes eronder bewegen op hetzelfde tempo.

Creational Patterns
De CellCollection class kan men ook als een singleton zien aangezien hier maar één van gemaakt is. Omdat ik mijn game in Unity2D maak is het erg lastig en zelfs onmogelijk om de juiste keywords te gebruiken om helemaal aan deze patterns te voldoen, daarom heb ik deze keywords vertaald naar de hierarchy.

Behavioral Patterns
De parent van elke cell gedraagd zich zoals een Memento, omdat wanneer een child cell destroyed is (SetActive(false)) hersteld de parent zijn child weer.

De Parents die helemaal boven aan staan en dus geen parent hebben. En de Children die helemaal onderstaan staan en dus geen children hebben fungeren als hun eigen Parent en Child. Aangezien Unity niet kan werken met Null Objects.

Toepassing van Design Patterns op Code

Oude Code (Zonder onderzoek Design patterns)

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

public class CellSplit : MonoBehaviour {

    public float splitTime = 5f;
    private float TimeLevel0 = 0.5f;
    private float TimeLevel1 = 1.0f;
    private float TimeLevel2 = 2.5f;
    private float TimeLevel3 = 3.5f;
    private float TimeLevel4 = 5.0f;
    private float TimeLevel5 = 8.0f;

    public float splitSize = 1f;
    private float CellSize0 = 0.125f;
    private float CellSize1 = 0.110f;
    private float CellSize2 = 0.090f;
    private float CellSize3 = 0.075f;
    private float CellSize4 = 0.060f;
    private float CellSize5 = 0.045f;

    private float X = 5f;
    private float Y = 8f;

    public int cellLevel = 0;
    public int cellKind = 0;

    public GameObject Cell;
    private GameObject[] CellArray;

    public bool CoreCell = false;

    private GameObject CCO;
    public GameObject ParentCell;



    //Initialization
    public void Awake () {
        CCO = GameObject.Find ("CellCollection");
```

```
        CellArray = new GameObject[2];

        if (CoreCell) {
            ParentCell = gameObject;
            StartSplit ();
        }
    }


    //Generate 2 cells below
    private void Split(){
        CellOneSplit ();
        CellTwoSplit ();
    }


    //Generate the first cell
    private void CellOneSplit(){
        //Create Cell 1
        Vector2 CellPos1 = new Vector2 (transform.position.x - (splitSi
ze * X), transform.position.y -  (splitSize * Y));
        GameObject Cell1 = GameObject.Instantiate (Cell, CellPos1, Quat
ernion.identity);

        //Set Parent
        Cell1.transform.SetParent(CCO.transform);

        //Set Cell properties
        CellSplit CS1 = Cell1.GetComponent<CellSplit>();
        CS1.cellLevel = cellLevel + 1;
        CS1.ParentCell = gameObject;
        CS1.cellKind = 1;

        //Start CellSplit
        CS1.StartSplit();
    }


    //Generate the second cell
    private void CellTwoSplit(){
        //Create Cell 2
        Vector2 CellPos2 = new Vector2 (transform.position.x + (splitSi
ze * X), transform.position.y -  (splitSize * Y));
        GameObject Cell2 = GameObject.Instantiate (Cell, CellPos2, Quat
ernion.identity);
```

```
        //Set Parent
        Cell2.transform.SetParent(CCO.transform);

        //Set Cell properties
        CellSplit CS2 = Cell2.GetComponent<CellSplit>();
        CS2.cellLevel = cellLevel + 1;
        CS2.ParentCell = gameObject;
        CS2.cellKind = 2;

        //Start CellSplit
        CS2.StartSplit();
    }


    //Start the splitting process
    public void StartSplit(){
        SetSplitValues ();
        StartCoroutine (Timer ());
    }


    //Set SplitTime and CellSize
    private void SetSplitValues(){
        switch (cellLevel){
        case 0: splitTime = TimeLevel0; splitSize = CellSize0;
            break;
        case 1: splitTime = TimeLevel1; splitSize = CellSize1;
            break;
        case 2: splitTime = TimeLevel2; splitSize = CellSize2;
            break;
        case 3: splitTime = TimeLevel3; splitSize = CellSize3;
            break;
        case 4: splitTime = TimeLevel4; splitSize = CellSize4;
            break;
        case 5: splitTime = TimeLevel5; splitSize = CellSize5;
            break;
        }

        transform.localScale = new Vector3 (splitSize, splitSize, split
Size);
    }


    //Timer before Splitting
    private IEnumerator Timer(){
        yield return new WaitForSeconds (splitTime);
```

```
        if (cellLevel < 5) {
            Split ();
        }
    }


    //Set timer for Single Splitting
    private IEnumerator SingleTimer(int Kind){
        yield return new WaitForSeconds (splitTime);
        if (Kind == 1) {
            CellOneSplit ();
        }
        if (Kind == 2) {
            CellTwoSplit ();
        }
    }


    //When Child has been destroyed
    public void Dead(int Kind){
        StartCoroutine (SingleTimer (Kind));
    }

}//CLASS
```

Nieuwe Code (Re-design van code-structuur)

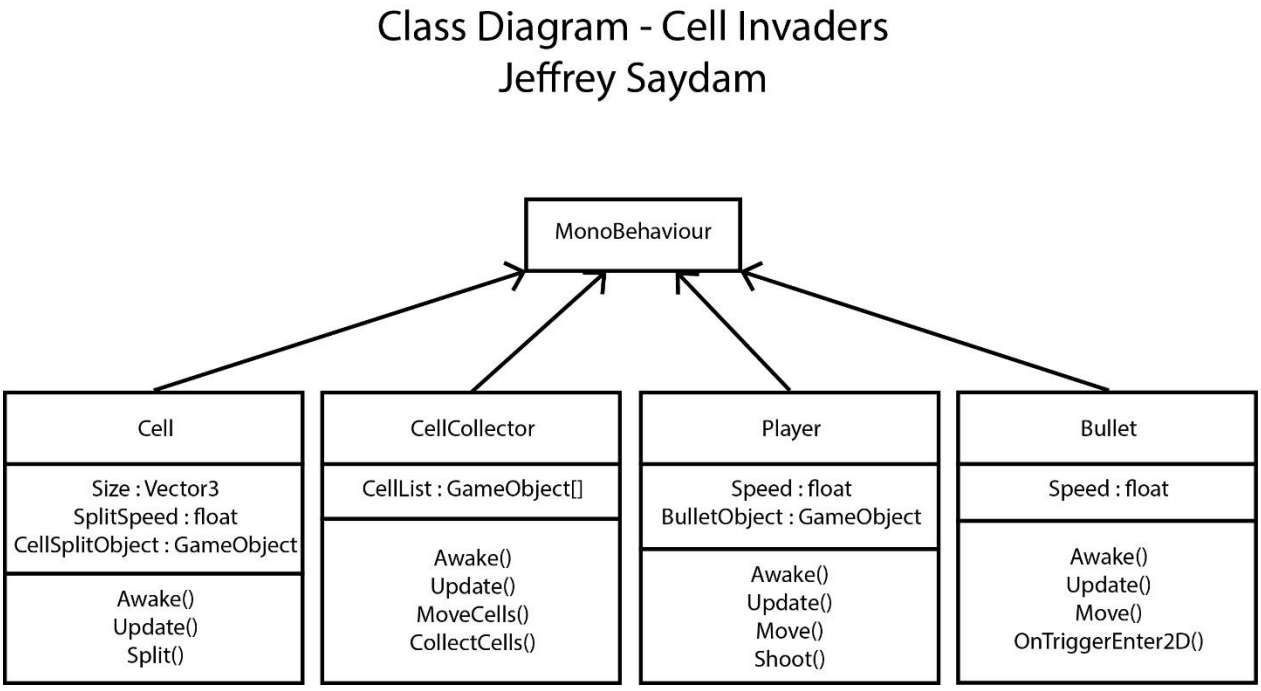
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CellSplitting : MonoBehaviour {

    public GameObject ParentCell;
    public GameObject ChildCell1;
    public GameObject ChildCell2;

    public float splitTime = 1f;

    public bool hasParent = true;
    public bool hasChildren = true;

    public bool inSplitTime1 = false;
    public bool inSplitTime2 = false;

    public bool FrozenCell = false;

    private SpriteRenderer SR;
    public Sprite CellNormal;
    public Sprite CellFrozen;


    //Initialization
    private void Awake () {
        //Components
        SR = GetComponent<SpriteRenderer>();

        //Security
        if (!hasParent) {
            ParentCell = gameObject;
        }
        if (!hasChildren) {
            ChildCell1 = gameObject;
            ChildCell2 = gameObject;
        }

        //Initialization
        if (hasParent) {
            ParentCell = transform.parent.gameObject;
        }
```

```
    if (hasChildren) {
        ChildCell1 = transform.GetChild (0).gameObject;
        ChildCell2 = transform.GetChild (1).gameObject;
    }
}


// Update is called once per frame
private void Update () {
    if (hasChildren) {
        CheckActive ();
        CheckFrozen ();
    }
}


//Checks if Cell1 or Cell2 has been destroyed
private void CheckActive(){
    if (!ChildCell1.activeSelf && !inSplitTime1) {
        StartCoroutine (SplitTime (1));
        inSplitTime1 = true;
    }
    if (!ChildCell2.activeSelf && !inSplitTime2) {
        StartCoroutine (SplitTime (2));
        inSplitTime2 = true;
    }
}


//Respawn the cell after a period of time
private IEnumerator SplitTime(int WhichCell){
    yield return new WaitForSeconds (splitTime);
    if (gameObject.activeSelf) {
        switch (WhichCell) {
        case 1:
            ChildCell1.SetActive (true);
            inSplitTime1 = false;
            break;
        case 2:
            ChildCell2.SetActive (true);
            inSplitTime2 = false;
            break;
        }
    }
}
```

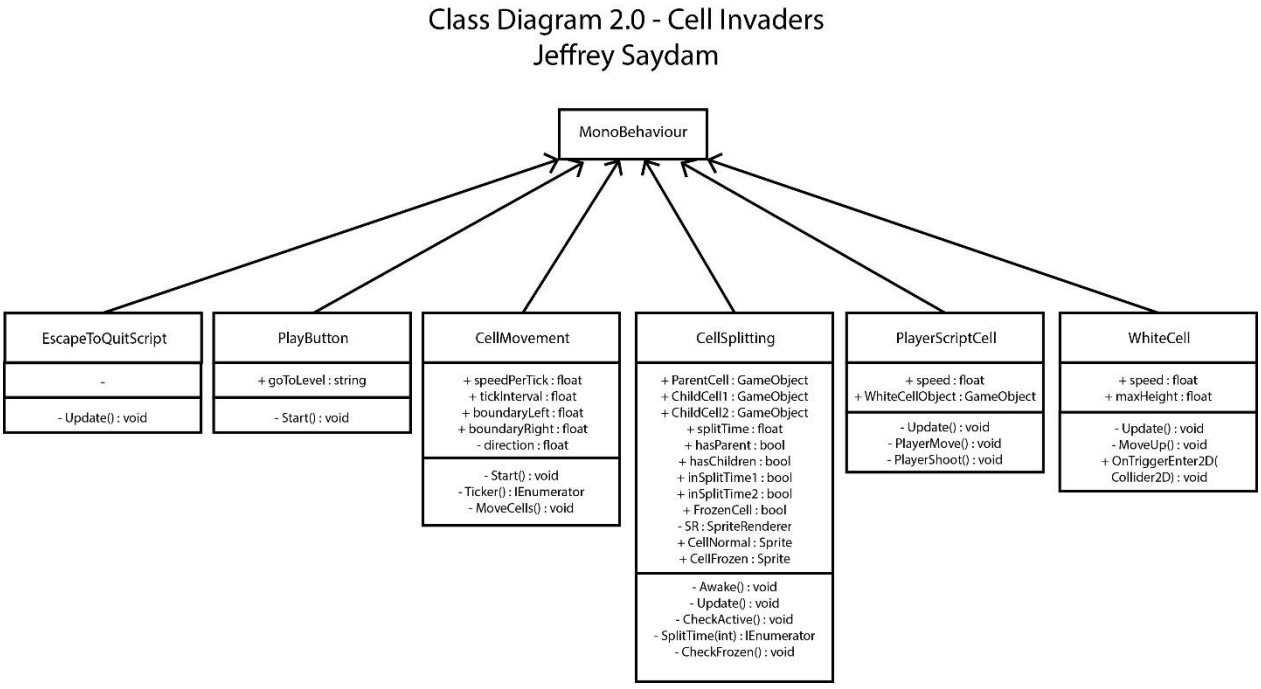
```
    private void CheckFrozen(){
        if (!ChildCell1.activeSelf && !ChildCell2.activeSelf) {
            SR.sprite = CellNormal;
            FrozenCell = false;
        }
        else {
            SR.sprite = CellFrozen;
            FrozenCell = true;
        }
    }

}//CLASS
```

Class Diagram Ontwikkeling

Class Diagram 1



Class Diagram 2



Bronnen

Art: Alles gemaakt door mij (Jeffrey Saydam)
Muziek: www.youtube.nl
Code: Alles gemaakt door mij (Jeffrey Saydam)