
# BUST-A-MOVE TWIST

KERNVAK GAME DEVELOPMENT – RETRO GAME

Kayleigh van der Veen & Lisa Perelaer

# INHOUDSOPGAVE

## WAAROM BUST-A-MOVE?

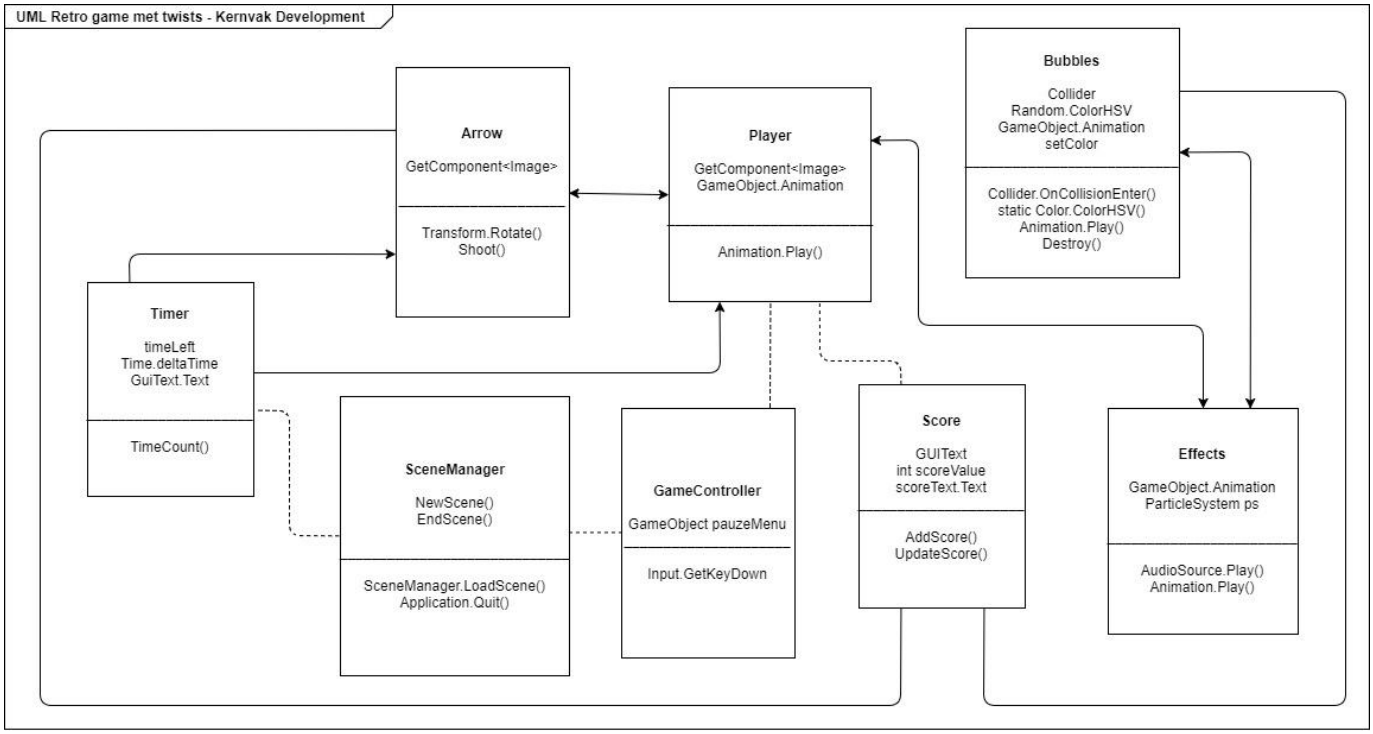
Wij hebben voor het spel Bust-A-Move gekozen omdat het voor ons beide een nostalgisch spel is. Daarbij vonden we het ook weer eens heel wat anders dan een shooter of simpelweg iets waar je doorheen rent (met het klassieke thema van een bewegend hoofdpersoon die op enemies afgaat).

Het leek ons ook interessant om te kijken hoe je met een grid werkt in Unity, omdat er niet standaard een optie is om een grid te maken. Achteraf hebben we er een beetje spijt van, omdat het zo nieuw voor ons was konden we niet zo goed focussen op de design patterns of de nieuwe lesstof toepassen.

We hebben er wel veel van geleerd, we zijn toch uit een soort comfortzone gestapt. Daarom zijn we alsnog redelijk blij met wat we hebben gemaakt.

## UITLEG UML

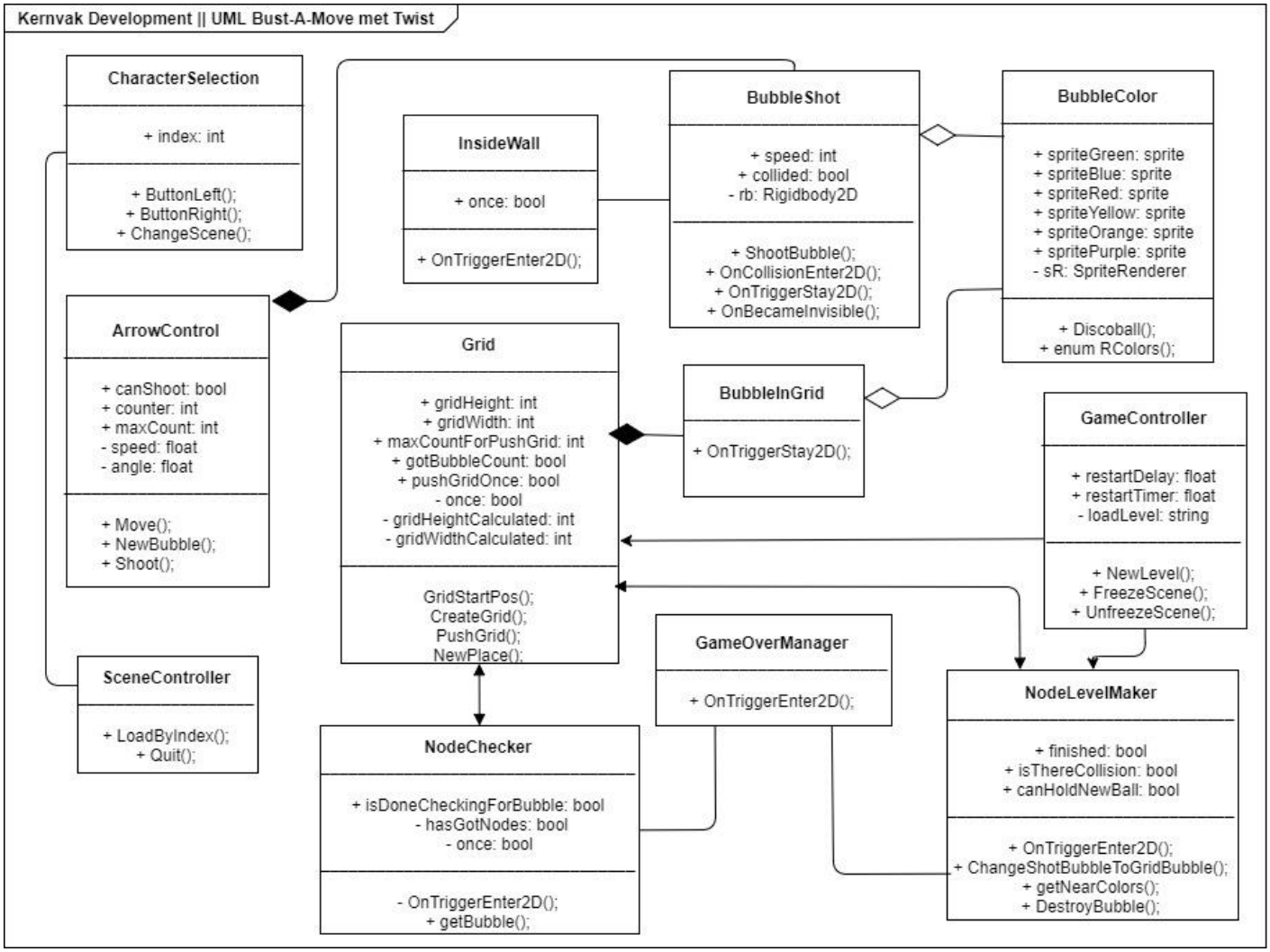
### CLASS DIAGRAM



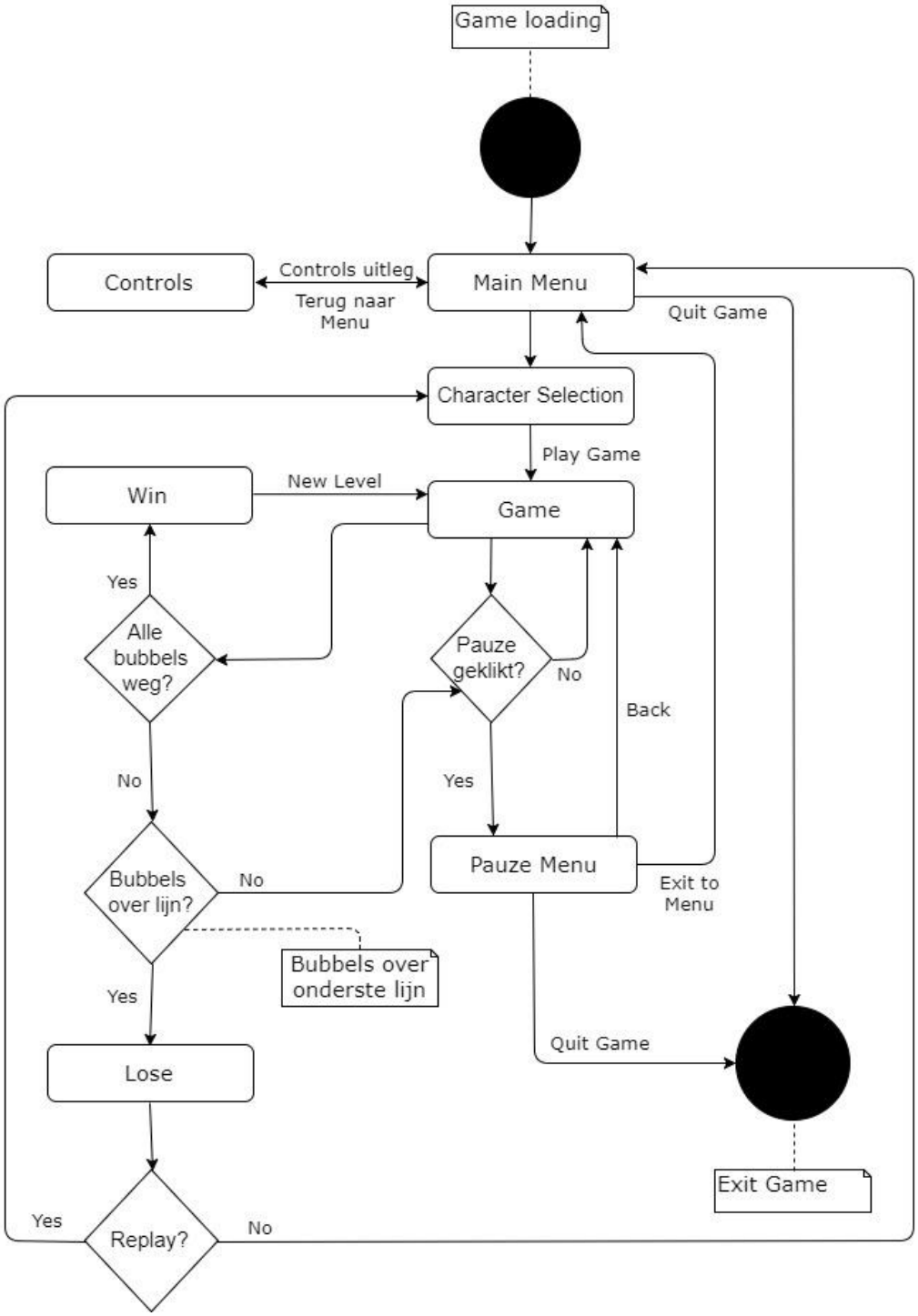

**Onze eerste class diagram van dit spel**

Onze eerste UML was eigenlijk te makkelik over nagedacht. We waren vooral bezig met nadenken over wat je namelijk ziet; effecten, een character class (hier nog player genoemd), score, een gamecontroller, een timer en een pijl. We zijn daarentegen nooit toegekomen aan effecten, score en timers omdat de bubbles afhankelijk waren van een grid.

Het uiteindelijke class diagram is de volgende:


Kernvak Development || UML Bust-A-Move met Twist
CharacterSelection
+ index: int
+ ButtonLeft();
+ ButtonRight();
+ ChangeScene();
InsideWall
+ once: bool
+ OnTriggerEnter2D();
BubbleShot
+ speed: int
+ collided: bool
- rb: Rigidbody2D
+ ShootBubble();
+ OnCollisionEnter2D();
+ OnTriggerStay2D();
+ OnBecameInvisible();
BubbleColor
+ spriteGreen: sprite
+ spriteBlue: sprite
+ spriteRed: sprite
+ spriteYellow: sprite
+ spriteOrange: sprite
+ spritePurple: sprite
- sR: SpriteRenderer
+ Discoball();
+ enum RColors();
ArrowControl
+ canShoot: bool
+ counter: int
+ maxCount: int
- speed: float
- angle: float
+ Move();
+ NewBubble();
+ Shoot();
Grid
+ gridHeight: int
+ gridWidth: int
+ maxCountForPushGrid: int
+ gotBubbleCount: bool
+ pushGridOnce: bool
- once: bool
- gridHeightCalculated: int
- gridWidthCalculated: int
GridStartPos();
CreateGrid();
PushGrid();
NewPlace();
BubbleInGrid
+ OnTriggerStay2D();
GameController
+ restartDelay: float
+ restartTimer: float
- loadLevel: string
+ NewLevel();
+ FreezeScene();
+ UnfreezeScene();
SceneController
+ LoadByIndex();
+ Quit();
GameOverManager
+ OnTriggerEnter2D();
NodeChecker
+ isDoneCheckingForBubble: bool
- hasGotNodes: bool
- once: bool
- OnTriggerEnter2D();
+ getBubble();
NodeLevelMaker
+ finished: bool
+ isThereCollision: bool
+ canHoldNewBall: bool
+ OnTriggerEnter2D();
+ ChangeShotBubbleToGridBubble();
+ getNearColors();
+ DestroyBubble();

## ACTIVITY DIAGRAM




Gelukkig iets makkelijker te zien en geen aanpassingen nodig gehad: de activity diagram. Het spreekt behoorlijk voor zich dus ik neem aan dat er geen verdere uitleg bij nodig is.

# PROCESS VAN HET MAKEN

## BEWEGING ARROW

Lisa (ik) heeft de arrow class geprogrammeerd. Dit duurde heel lang omdat ik niet snapte waarom mijn code niet werkte, en elke keer dat ik iets anders probeerde werkte het toch niet.

Ik heb het alleen in het begin stadium naar gitub gepusht, helaas niet wat er tussenin zat want dat was wel heftig, maar om een klein stukje er nog van te laten zien:

```
15  -       public void Move(float axis) {
16  -           if (transform.eulerAngles.z <= -90) {
17  -               if (axis > 0) {
18  -                   axis = 0;
19  -               }
20  -           }
21  -           if (transform.eulerAngles.z >= 90) {
22  -               if (axis < 0) {
23  -                   axis = 0;
24  -               }
25  -           }
26  -           Debug.DrawRay(shootPoint.position, transform.up * 5, Color.red);
27  -           Debug.Log(transform.eulerAngles.z);
28  -           this.transform.Rotate(0.0f, 0.0f, -axis * speed);
```

Vóór de tip van Valentijn om Mathf.Clamp te gebruiken om te zorgen dat de pijl niet 360 graden kon draaien, had ik een grote if statement die de angles moest checken. Voordat ik de euler angles checkte gebruikte ik transform.rotation, alleen is die, als ik het me goed herinner, afhankelijk van welke kant je begint met draaien.

Uiteindelijk werkte het dat de pijl tussen bepaalde waarden bleef, maar als hij de grenswaarde bereikte, dan werkte de pijl niet meer. Daarbij checkte ik per Input waarde of de pijl dan naar links wél of niet mocht, of juist naar rechts wel of niet mocht. Maar dat leek niet te helpen, en ook al printte ik al mijn angles en stond die bijvoorbeeld op 270 graden – (in de if stond dan 'if eulerangles.z >= 270) – dan moest er de andere kant op kunnen worden bewogen. De uiteindelijke oplossing is deze:

```
private void Move() {
            angle += Input.GetAxisRaw("Horizontal") * speed;
            angle = Mathf.RoundToInt(angle);
            angle = Mathf.Clamp(angle, -85, 85);
            transform.rotation = Quaternion.Euler(0, 0, angle);

        }
```

Door het gebruik van clamp kon ik de angle wel afbakenen.

Een ander 'probleem' wat je ook kan zien in deze code is dat je in bust-a-move per graad beweegt, dus er zit geen smoothing tussen. Met de standaard rotate gebeurde dit wel, dus ook dit hebben we aangepast.

## GRID

Het grid is het moeilijkste geweest. Kayleigh heeft dit zo'n 2 a 3 weken proberen te onderzoeken, maar het was allemaal of heel vaag, of heel wiskundig en snapten we er niks van.

Het volgende is het grid waar we mee begonnen.

```
public Transform BubblePrefab;
public int gridHeight = 14;
public int gridWidth = 6;

float bubbleHeight = 1.0f;
float bubbleWidth = 1.0f;

public float distance = 0.0f;

Vector3 startPos;       // defines where first bubble will be placed

// UNITY FUNCTIONS ----------------------------------------
void Start() {
    AddDistance();      // adding distance offset
    GridStartPos();     // calculating the start position
    CreateGrid();       // create the grid
}


// FUNCTIONS ----------------------------------------
void AddDistance() {
    bubbleHeight += bubbleHeight * distance;
    bubbleWidth += bubbleWidth * distance;
}

void GridStartPos() {
    float offset = 0;

    if (gridHeight / 2 % 2 != 0) {
        offset = bubbleWidth / 2;
    }

    float x = -bubbleWidth * (gridWidth / 2); //- offset
    float z = bubbleHeight * 0.75f * (gridHeight / 2);

    startPos = new Vector3(x, 0, z);
}
```

```
Vector3 WorldPos(Vector2 gridPos) {     // function that translates the coordinates
    float offset = 0;

    if (gridPos.y % 2 != 0) { // if not devideable by two
        offset = gridWidth / 2;
    }

    float x = startPos.x + gridPos.x * bubbleWidth; //+ offset
    float z = startPos.z - gridPos.y * bubbleHeight * 0.75f;

    return new Vector3(x, 0, z);
}
```

```
void CreateGrid() {
    for (int y = 0; y < gridHeight; y++) {
        for (int x = 0; x < gridWidth; x++) {
            Transform bubble = Instantiate(BubblePrefab) as Transform;
            Vector2 gridPos = new Vector2(x, y);    // stores coordinates of the bubbles
            bubble.position = WorldPos(gridPos);    // translates 2D coordinates into 3D world
            bubble.parent = this.transform;         // stores bubbles in parent so hierarchy will stay clear
            bubble.name = "Bubble" + x + "|" + y;   // position of the bubbles
        }
    }
}
```

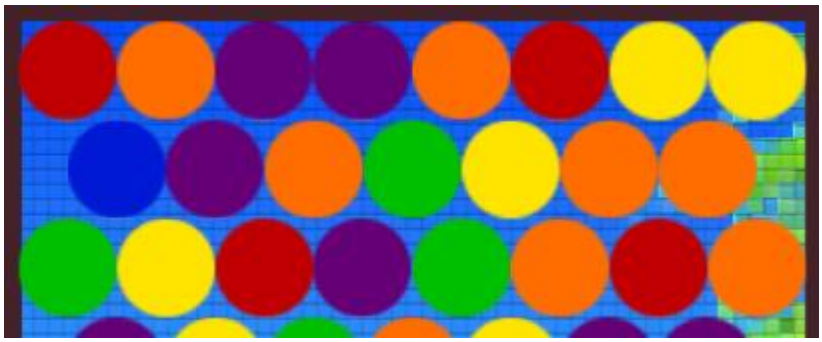
Daarna zijn we samen gaan werken om het grid goed te krijgen, omdat we beide het niet alleen konden.

De uiteindelijke class is nog groter dus het is handiger als u het er even bij pakt.

We hebben uiteindelijk een grid weten te maken die een prefab spawnt van een node.

Het grid start bij de transform van het gameobject waar het script op zit, en de *pivot* ervan is in de linker-onderhoek. Vervolgens kan je in de inspector aangeven hoe breed en lang het grid moet zijn. Van daaruit rekent het script welke posities dat precies zijn. (Als het grid 6 lang is en hij begint bij -4, dan moet de laatste lijn beginnen bij y = 2).

Vervolgens wordt het grid gecreeërd in de functie **CreateGrid();** Hier wordt via 2 forloops de x en y van elke node bepaald. Als de rij even is dan is er een offset :



In de 2e rij moet dan ook 1 bubble minder worden aangemaakt, anders gaat die over de muur heen.

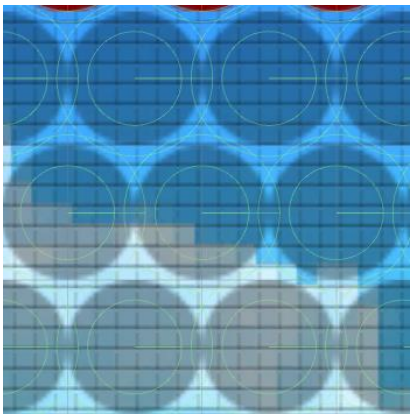
Dit wordt in diezelfde functie nog geregeld.

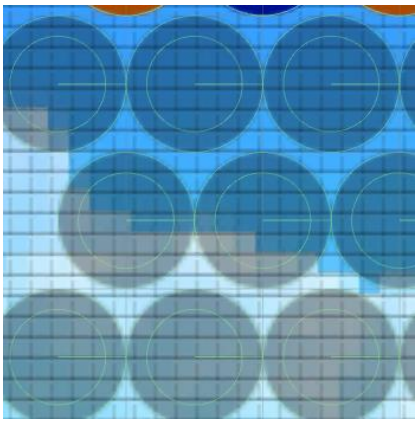
**CreateGrid** noemt **NewPlace** aan, waarin de x en y van de 2 for loops mee worden genomen. Er wordt vervolgens een node neergezet. Daarna wordt er een nieuwe vector aangemaakt die x en y bevatten en gaat die nog door een andere functie heen, namelijk **SetPos.** Deze zorgt voor de offset.

Daarna zie je een gekke staan: voor zowel x als y komt er 0.5f bij. Dit is omdat de nodes en alle ballen hun pivot in het midden staan. Om te zorgen dat elke bal een normale transform positie heeft (en niet teveel getallen achter de decimaal) komen deze 2 erbij. Daarna word de node's parent het grid en wordt er een naam gebaseerd op de transform aan gegeven zodat je het makkelijker kan vinden als je het nodig hebt.

## NODES

Er zijn 2 scripts voor de nodes. Dit komt omdat om verschillende doelen te bereiken, de scripts afhankelijk waren van de colliders op de nodes zelf. Omdat de rijen te maken hebben met een offset, raken de colliders elkaar alleen zijdelings.





Wanneer het script start, zijn de buitenste colliders zoals in het eerste plaatje. Elke node maakt via de collision een lijst met welke (en hoeveel dus) nodes er om hem heen liggen. Omdat later in het spel de geschoten bubbels nauwkeuriger snappen als deze collider gelijk is met de plek waar de bal in kan snappen, wordt de collider kleiner gemaakt wanneer deze lijst is gevuld.

De binnenste collider werkt voor de bubbles die je in de scene kan slepen om levels te maken. Wanneer deze colliders groot zijn, moet je je bubbels ontzettend nauwkeurig de scene in slepen, anders snapt hij naar degene

wiens script het laatst wordt uitgevoerd. (Immers de bubble zit in beide triggers!) Door deze kleiner te maken snappen ze een stuk nauwkeuriger. Hierna worden deze colliders niet meer gebruikt, maar de grote.

Omdat er zovel processen zijn die vrijwel tegelijk worden uitgevoerd, staan er zoveel booleans in beide scripts. Voor vrijwel elke kleine check moet er worden gezorgd dat iets maar 1 keer wordt uitgevoerd, of dat iets in een ander script pas wordt uitgevoerd is als een specifiek stukje klaar is. En daarom zijn er ook zoveel public variables, want de scripts moeten die referenties hebben.

Een stukje in NodeLevelMaker waarop we trots zijn is **ChangeShotBubbleToGridBubble**, waarin we een geschoten bubbel (die allerlei eigen settings heeft m.b.t. rigidbody en script) omzetten in een bubbel die in het grid blijft hangen. De Rigidbody moet zo kinematic worden, de force waarmee hij is afgevuurd moet eraf, zijn tag moet veranderen zodat de pijl weer weet wanneer er mag worden geschoten (als je maar kan blijven afvuren dan gaan nog niet gesnapte ballen van elkaar af bouncen).

De reden dat deze tag **niet** hetzelfde is als de ballen die in de scene van tevoren zijn gesleept, is het checken van hun plek; anders worden ze dubbel gesnapt aan het grid (namelijk ook via NodeLevelMaker OnTriggerEnter2D met 'BubbleInScene' . Dit gaf problemen.

Veel voorkomende problemen waren dat ballen op al gevulde plekken gingen zitten, maar vooral proberen goed te krijgen in welke volgordes dingen gebeurden (ookal hebben verschillende scripts dus verantwoordelijkheden die bij start gebeuren, of elke frame, maar pas als er een aantal dingen zijn geregeld!)

In eerste instantie hadden we de pivot van de bubbles op linksonder, maar dan ging de OnCollisionEnter wat minder fijn.

Dit heeft ons allemaal heel erg lang gekost om goed te krijgen.

## ONOPGELOSTE PROBLEMEN

Misschien zagen jullie het al in het filmpje, maar nog steeds hebben we last van bubbles die verdwijnen wanneer ze horen te snappen, maar ook bubbles van dezelfde kleur die kapot moeten gaan en er vervolgens 1 van een andere kleur meenemen.

Wij hebben dit zo proberen op te lossen:

```
public void getNearColors(GameObject changeBub) {
            if (!bubblesSameColorList.Contains(changeBub)) {
                bubblesSameColorList.Add(changeBub);
            }

            foreach (GameObject bubble in bubblesAroundList) {
                if (!bubblesSameColorList.Contains(bubble)) {
                    BubbleColor colorScript = bubble.GetComponent<BubbleColor>();
                    if (colorScript.rColors ==
changeBub.GetComponent<BubbleColor>().rColors) {
                        bubblesSameColorList.Add(bubble);
                    }
                }
            }
        }

        public void DestroyBubble() {
            if (bubblesSameColorList.Count >= 2) {
                foreach (GameObject bubble in bubblesSameColorList) {
                    Destroy(bubble);
                }
            }
        }
```

In deze lijst lijken ook alleen bubbles van dezelfde kleur te komen, maar in de game gaan er kleuren mee. We hebben dit niet kunnen oplossen.

## GITHUB

Op het laatst zijn we heel veel tijd kwijt geweest aan problemen met github.

Het programma mergde dan dingen fout; soms hadden we opeens code dubbel. Soms waren aanpassingen weer gereset. De scenes gingen niet goed mee en toen hebben we lang in de code zitten snuffelen om te zoeken waarom het opeens (wéér) niet werkte, net nadat we het hadden gemaakt en het toen werkte. Dan bleken er geen tags te zitten op de prefabs, waren de pivots weer gereset naar linksonder in plaats van gecentreerd, waren de aanpassing aan de prefabs niet opgeslagen..

En een enkele keer kregen we opeens rare tekst en cijfers in onze code:

```
16
17          public RColors rColors;
18
            -<<<<<<< HEAD
19          +
20          private SpriteRenderer spriteRenderer;
21
            -       private void Update() {
            -=======
22          +
23
24          private void Start() {
            ->>>>>>> f4ff2aec87308f1447a97702fbaf8673b5a52c95
25              DiscoBal();
26          }
```

## DESIGN PATTERNS

Wij hebben uiteindelijk geen design patterns toegepast. We waren zolang bezig met het grid en al die kleine foutjes dat we er niet aan toegekomen zijn.

# BRONNEN

### INCLUDING WORST EXPLANATIONS IN HISTORY, MAYBE EVER

### YOUTUBE

Bubble Shooter Asset:

https://www.youtube.com/watch?v=-B_WB4OzbiA

Bubble Shooter Game Source Code Unity:

https://www.youtube.com/watch?v=EKRQeIm5UIo

https://www.youtube.com/watch?v=td3O1tkbqYQ&t=5s

## OTHER

https://www.redblobgames.com/grids/hexagons/

http://csharphelper.com/blog/2015/10/draw-a-hexagonal-grid-in-c/

https://www.codeproject.com/Articles/14948/Hexagonal-grid-for-games-and-other-projects-Part

https://yal.cc/understanding-isometric-grids/