BubleTroubleTwisted

Bas de Koningh - 3028315

Kernmodule Game Dev

19-09-2019

Referentie/Inspiratie:

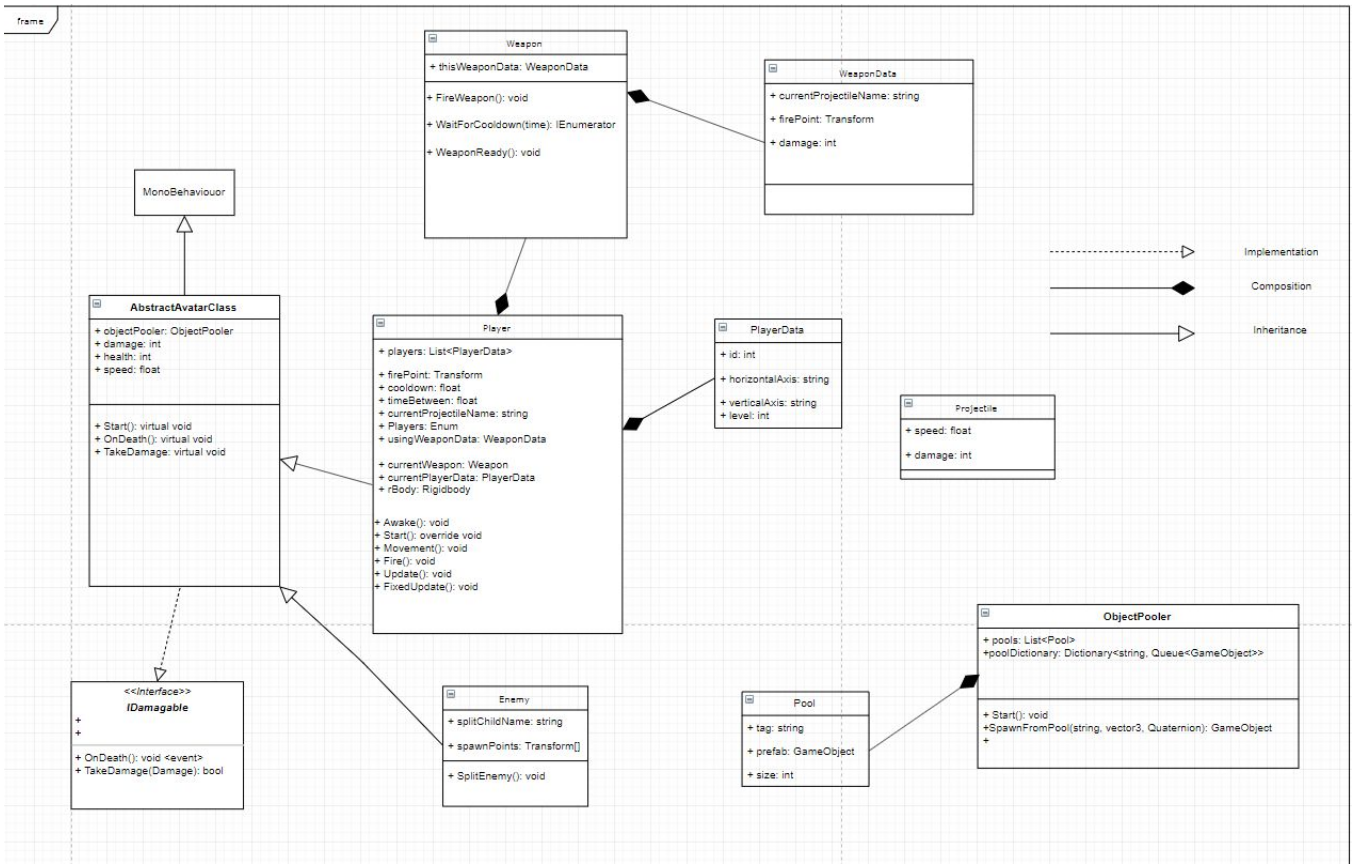
Opdracht: Retro Game met een twist
Retro game: Bubble Trouble
Twist: 3D

Het spel werkt als volgt:
Je speelt als een kleine capsule die alleen of met meerdere spelers speelt.
Het doel is om alle stuiterende ballen kapot te maken en verder naar volgende levels te komen.

Je schiet een straal blokjes omhoog die als ze de ballen raken de ballen in 2e splitsen en deze worden kleiner des te vaker je ze split totdat ze helemaal kapot zijn.

Ik ben begonnen met het uitdenken van hoe ik de basis van de speler en ballen wilde gaan maken. Hiervoor heb ik gekozen om een AbstractAvatarBase class te maken waarvan zowel de ballen als de spelers erven. Zowel de speler als de ball hebben bepaalde basis attributen die overeen komen. Ik heb ook nagedacht over hoe ik de blokjes die de spelers afschieten het beste kon laten “spawnen”. Ik heb het begin opgezet met playerdata waarmee er eenvoudig meerdere spelers kunnen spelen. Omdat er dus meerdere spelers in het spel kunnen spelen en deze allemaal blokjes schieten heb ik ervoor gekozen om een ObjectPooler in te zetten. Dit zorgt ervoor dat ik aan het begin van het spel wat geheugen van de computer inzet om de cpu performance later in het speler zo licht mogelijk te houden.

Dit was mijn eerste versie van de UML:



Ik ben later gaan nadenken over hoe ik bepaalde events en references over en weer willde gaan sturen. Ik heb ervoor gekozen om de managers zo goed als gelukt is uit elkaar te trekken zodat elke class doet waar die voor staat. Voor het aanroepen van methodes over verschillende classes heen heb ik ervoor gekozen om een EventManager te bouwen. De EventManager is een Static class met static methods waar je dus vanauit elke class bij kunt. Wat de EventManager doet en moet doen is ongeveer hetzelfde als met een delegate: je subscribed(AddHandler

method in EventManager) je unsubscribed(RemoveHandlers method in Eventmanager) en callback(BroadCast method in EventManager). Zo zag mijn EventManager eruit:

```
public enum EVENT { gameUpdateEvent, reloadGame, initializeGame, saveGame, selectGame }; // ... Other events

public static class EventManager

{

    #region Handling events using System.Action()

    // Stores the delegates that get called when an event is fired

    private static Dictionary<EVENT, Action> eventTable

                = new Dictionary<EVENT, Action>();


    // Adds a delegate to get called for a specific event

    public static void AddHandler(EVENT evnt, Action action)

    {

        if (!eventTable.ContainsKey(evnt)) eventTable[evnt] = action;

        else eventTable[evnt] += action;

    }


    // Fires the event

    public static void Broadcast(EVENT evnt)

    {

        if (eventTable[evnt] != null) eventTable[evnt]();

    }

    #endregion

}
```

Na itereren en meer te weten te komen over Generics ben ik gaan kijken hoe ik dit kan implenteren op classes die ik dan over andere projecten heen ook 100% kan hergebruiken. De

EventManagerGeneric<T> en de GenericSingleton<T, A> zijn de 2 classes die helemaal 100% hergebruikt kunnen worden. Omdat de normale EventManager geen parameters mee kon sturen heb ik ervoor gekozen om hem generic te maken zodat je elk type mee kunt sturen en je ook aan 1 EVENT meerdere soorten types kunt toevoegen. Zo ziet de EventManagerGeneric<T> er nu uit:

```
/// <summary>

/// Generic event management/ this class is re-usable in every project

/// </summary>

/// <typeparam name="T"></typeparam>

public static class EventManagerGen<T>

{

    public delegate void GenericDelegate<A>(T c);



    // Stores the delegates that get called when an event is fired

    static Dictionary<EVENT, GenericDelegate<T>> genericEventTable = new
Dictionary<EVENT, GenericDelegate<T>>();



    // Adds a delegate to get called for a specific event

    public static void AddHandler(EVENT evnt, GenericDelegate<T> action)

    {

        if (!genericEventTable.ContainsKey(evnt)) genericEventTable[evnt] = action;

        else genericEventTable[evnt] += action;

    }



    // Fires the event

    public static void BroadCast(EVENT evnt, T c)

    {

        if (genericEventTable[evnt] != null) genericEventTable[evnt](c);
```

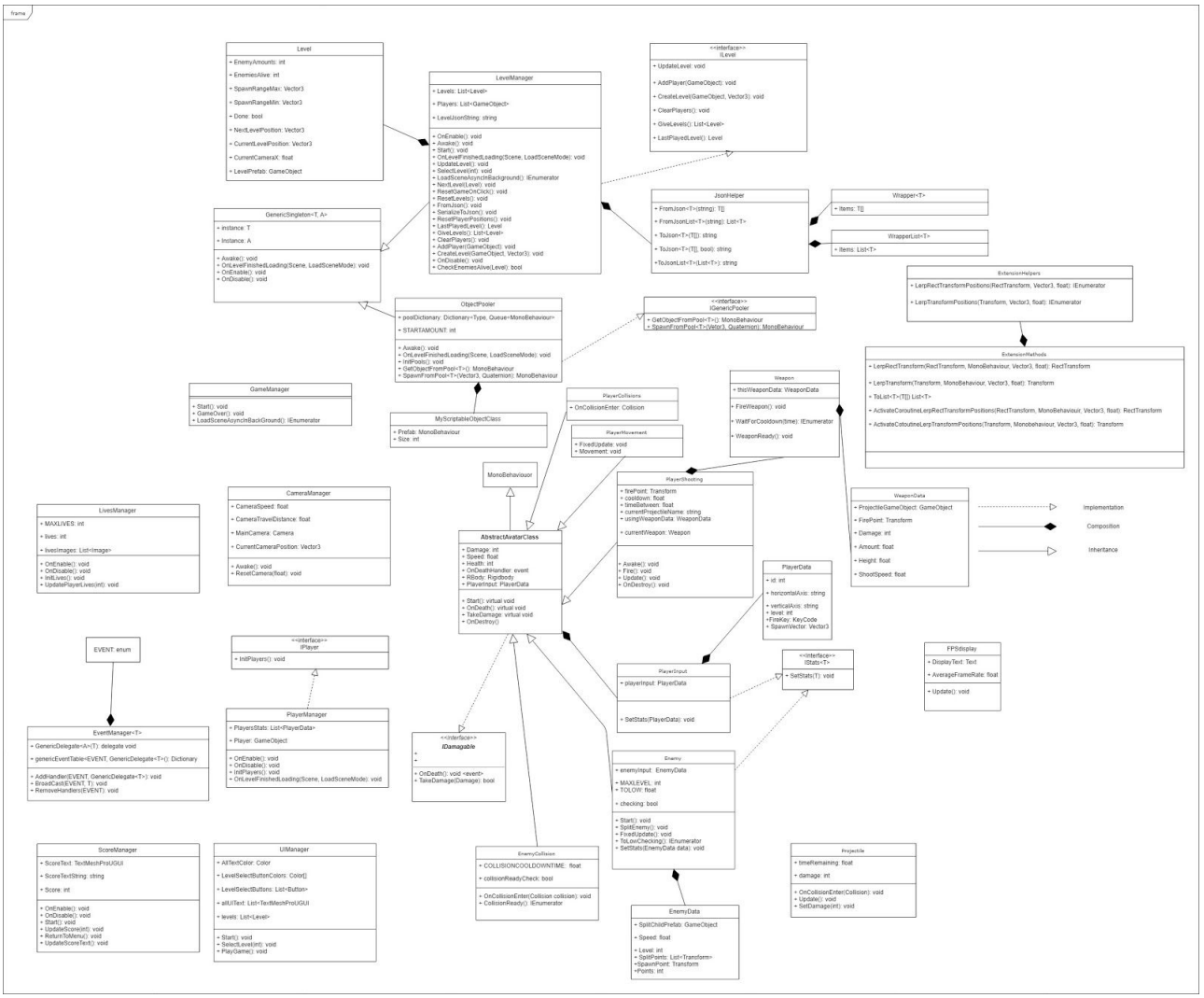
```
    }

    //Un-Subscribes the listeners to the event
    public static void RemoveHandlers(EVENT evnt)
    {
        if (!genericEventTable.ContainsKey(evnt))
        {
            return;
        }
        else
        {
            foreach (KeyValuePair<EVENT, GenericDelegate<T>> _delegate in genericEventTable)
            {
                genericEventTable[evnt] -= _delegate.Value;
            }
        }
    }
}
```

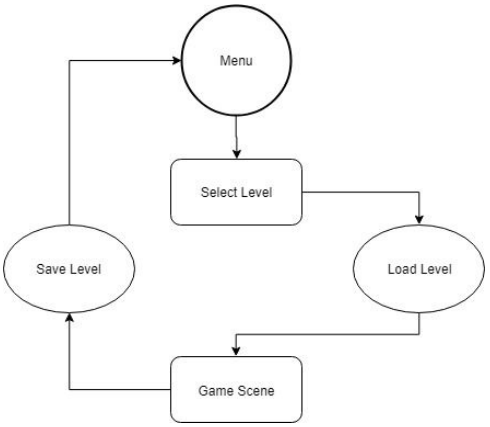
Voor mijn spel wilde ik graag iets maken waar alle levels in 1 scene zijn en deze als je level N gehaald hebt en je sluit de applicatie af opslaat dat je bij N was gebleven. Hiervoor heb ik een soort level Select menu gemaakt. Ik heb ervoor gekozen om dit met Json serialization te doen omdat ik hier persoonlijk bekend mee ben ik en zelf Json fijn vindt werken. Dit managen serialiseren en uitlezen gebeurd allemaal in de LevelManager class. Dit is daarom ook de grootste class die ik in mijn project heb staan. (Ongeveer 300 regels) Als de speler dood gaat wordt het spel opgeslagen en wordt de data geserialiseerd naar Json, volgende keer dat de speler het spel opstart wordt deze data eerst uitgelezen en daarna kan de speler kiezen welk level hij wilt spelen.

Hierboven noemde ik al de GenericSingleton<T, A>, ik heb ervoor gekozen om deze class te maken omdat ik meerdere singletons wil gebruiken en deze allemaal van de GenericSingleton class kunnen erven. Dit scheelt weer overbodig veel dezelfde code. Ik heb er ook voor gekozen om deze class niet alleen een T maar ook een A mee te geven. T refereert naar Component en

A naar een interface. Hiermee zorgt ie er gelijk voor dat als de singleton instance wordt opgevraagd in een andere class om een method te callen, dat niet de monobehaviour wordt aangesproken maar de interface. Dit zorgt ervoor dat de singleton beter beveiligd is en dat gebruikers niet per ongeluk iets als Destroy() kunnen callen.

Dit is de final version van mijn UML:





Dit is het activity diagram van de gameflow:

KLAD:

Iterate OOP design voor vandaag deadline 1700

Wat ik nu heb:

-Meerdere spelers

-Blokjes die met parabolic curve omhoog schieten

-Ballen die als je ze raakt in 2e splitsen

-Meerdere levels

-ObjectPooler global op 1 plek

-EventManager die static delegates bewaard en die functionaliteit bied om ook met action events te werken.

-LevelManager die bijhoud wanneer de speler verder mag (als alle enemies p level verslagen zijn)

-Veel oude code die uitgecomment staat

-Slordige plaatsing van: private / public fields en properties

Wat ik nog wil:

-Game over gaan als je levens op zijn

-Systeem voor het bijhouden van levens en score (punten)

-Meerdere levels

-Meer variatie per level (verschillende enemy types / power ups)

-Code opschonen en refactoren waar nodig

-Beetje UI

-Beetje particle systems en post processing om het er leuker uit te laten zien.

-Game won scherm

-Achtergrond

EXTRA:
-High score system
-Clean UI
-Editor Window voor m'n object pooler

-Sound effects

Feedback:

-ballen meer bewegen (links / rechts)
-bullet straal doortrekken naar boven
-power ups

TO-DO:

- In menu scene een level selection screen
    - hier moet de levelmanager uitgelezen worden en de speler krijgt voor elk level een vakje met een cijfer te zien.
    - als de speler op een vakje klikt dan wordt dit level gepakt en hier de done : bool = false gezet.
    - dan laad de game scene in en begint de speler bij het gekozen level.
    - in de levelmanager moet ipv alleen te zoeken naar het eerst gevonden level waarvan de done : bool op false staat ook als de speler het huidige level weet te behalen er gezocht worden naar de volgende index.
        - ■

BUGS:

High
-Als je het level hebt voltooid en je naar het volgende level zou moeten gaan. Dan verplaatst de camera en wordt de level manager geupdate alleen de spelers verplaatsen niet mee.

```
/*

//TO DO:

* BUG analysis: als ik op de reset game knop druk dan wordt het event OnGameOver gecalled
* die ervoor zorgt dat de scene opnieuw opstart. Op het moment dat de scene opnieuw is geladen
* krijg ik meerdere nullreferenties die alleemaal erop neer komen dat bepaalde instanties niet meer
* te bereiken zijn. De objecten referen naar een oude "versie" van de instanties en omdat het level
* herladen is zijn er nieuwe instanties. Deze instanties worden niet geupdate bij de objecten. Dit zorgt
* voor een nullreferentie.
```

```
        *
        * Misschien kan het zijn dat omdat ik mijn scene herlaad en ik bepaalde events in mijn EventManager
        * nog niet un-subscribe deze conflicten en oude referenties blijven zoeken.
        *
        * Oplossingen:
        * --Kijken of ik de objecten kan laten updaten zodat deze de nieuwe versie van de instanties krijgen.
        * --Kijken of als ik ipv het herladen van dezelfde scene eerst terug kan gaan naar het menu en dan
        * opnieuw het spel in kan laden het probleem zich nog voordoet.
        * --De resetGame veranderen naar dat ipv de scene opnieuw wordt geladen alleen alle data die gereset
        * zou moeten worden resetten. Dit zorgt ervoor dat er geen nieuwere versie van de instanties gemaakt
        * kan worden.
        */
    /*(OPTIONAL)
     * Idee: Misschien is het leuk en handig om voor mijn levels een state machine
     * te bouwen die checkt: OnLevelEntered() OnLevelUpdated() OnLevelExited()
     * dit zorgt ervoor dat ik meer overzicht krijg over waar de speler zich bevindt
     * en over hoe ik de levels kan managen.
     */
```