



Unity Developer Flotsam

Realisatie document

Bachelor in Toegepaste Informatica -
Application Development Werktraject

Johan Torfs

Academiejaar 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

INHOUDSTAFEL

1	INLEIDING.....	4
2	INLEIDENDE CONCEPTEN.....	5
2.1	Programma's en tools	5
2.2	Belangrijke termen en concepten.....	5
2.2.1	Townheart	5
2.2.2	'Landmark'	5
2.2.3	'Drifter'.....	6
2.2.4	'Walkway'	6
2.2.5	'Overlay'	7
2.2.6	Upgraden van gebouwen	7
2.2.7	'CursorManager' en 'CursorProperties' klasse.....	8
2.2.8	'EnergyPoles'	10
3	PROJECTFASES.....	11
3.1	Bestaande bugs en problemen in het spel.....	11
3.1.1	'Difter' zit vast op 'walkway'	11
3.1.2	Bouwen op 'walkways' in opbouw, geeft problemen.....	12
3.1.3	'SeedContainer' kan niet afgebroken worden	13
3.1.4	Een andere drifter vervoert de materialen.....	13
3.1.5	Blokkeringsregio van de Townheart motor	14
3.1.6	Het afbreken van 'walkway' segmenten	14
3.2	Upgraden van 'walkways'	15
3.2.1	'Buildable'-, 'WalkwaySegment'- en 'WalkwayPonton'-klasse	15
3.2.2	'BuildableProperties'-bestand in de upgrade van walkways	16
3.2.3	De upgrade-methode van de WalkwaySegment- en WalkwayPonton-klassen	16
3.2.4	Gratis 'walkways'	17
3.3	'Architect overlay'	17
3.3.1	'Ups-and-downs' in het implementeren van de feature 'Architect Overlay' ..	18
3.3.2	Het ontwikkelen van de 'Architect Overlay'.....	18
3.3.3	De 'MoveBuildableCursorProperties' klasse	18
3.3.4	StoredBuildables 'dictionary'	21
3.3.5	ArchitectBuildMenu.....	21
3.3.6	Visuele 'overlay'	22
4	CONCLUSIE	23

1 INLEIDING

Dit realisatiedocument is opgesteld als een samenvatting van mijn stage-ervaring bij Pajama Llama Games. Tijdens mijn stageperiode heb ik de kans gekregen om te werken aan een opwindend project: het uitbreiden van het constructiesysteem in hun populaire spel genaamd Flotsam. In dit document zal ik de belangrijkste doelstellingen, uitdagingen en de behaalde resultaten van mijn stage bij Pajama Llama Games bespreken.

Gedurende mijn stageperiode heb ik me gericht op het identificeren van gebieden waar het constructiesysteem verbeterd kon worden. Ik heb nauw samengewerkt met het ontwikkelingsteam van Pajama Llama Games om nieuwe functies en mechanics te ontwerpen die zouden zorgen voor een intuïtievere en bevredigendere bouwervaring voor de spelers. Daarnaast heb ik ook gekeken naar manieren om de prestaties en stabiliteit van het constructiesysteem te optimaliseren.

Tijdens het uitvoeren van mijn taken kwam ik verschillende uitdagingen tegen, zoals het balanceren van de complexiteit van het constructiesysteem met de toegankelijkheid voor nieuwe spelers. Ik moest rekening houden met de bestaande gameplay-mechanismen en ervoor zorgen dat de nieuwe functies naadloos kunnen integreren in het spel. Met behulp van iteratieve ontwikkeling en feedback van het team kon ik deze uitdagingen succesvol overwinnen.

Het resultaat van mijn stage bij Pajama Llama Games is een verbeterd constructiesysteem dat spelers in staat stelt om creatievere en geavanceerdere structuren te bouwen in Flotsam. Nieuwe functies zoals het upgraden van de 'walkways' en de 'Architect overlay' zorgen namelijk voor een verhoogde betrokkenheid en tevredenheid onder de spelers. Bovendien hebben de optimalisaties ervoor gezorgd dat het constructiesysteem soepel en stabiel functioneert, zelfs bij complexe constructies.

Meer informatie over mijn stagebedrijf, Pajama Llama Games, en over hun bekroonde spel Flotsam, kan gevonden worden in het 'Plan van aanpak'-document, opgesteld voor deze stageopdracht.

2 INLEIDENDE CONCEPTEN

In dit hoofdstuk wordt er eerst meer uitleg gegeven over de gebruikte programma's binnen het bedrijf. Er worden ook enkele belangrijke termen uitgelegd. Deze termen zijn specifiek binnen Flotsam en zullen later in dit document gebruikt worden om geavanceerdere concepten uit te leggen.

2.1 Programma's en tools

Voor de ontwikkeling van Flotsam worden er veel verschillende programma's gebruikt. Het spel zelf wordt gemaakt in Unity (zie <https://unity.com>). Dit is een game-engine, een programma met als specifiek doel het ontwikkelen van games.

Daarnaast zijn er voor het ontwikkelen van een spel nog tal van andere programma's nodig.

- Voor het maken van 3d-modellen en animaties, wordt er bij Pajama Llama Games gebruik gemaakt van Blender (zie <https://www.blender.org>).
- Om het gedrag te bepalen van alle objecten in het spel moet er ook code geschreven worden. Dit gebeurt in C# en de editor die hiervoor binnen het bedrijf gebruikt wordt is Visual Studio (zie <https://visualstudio.microsoft.com>).
- Voor 'version control'¹ wordt er gebruik gemaakt van Bitbucket (zie <https://bitbucket.org>) en SourceTree (zie <https://www.sourcetreeapp.com>).
- Voor projectmanagement wordt Notion gebruikt (zie <https://notion.so>).
- Voor het opvolgen en rapporteren van bugs, gebruikt het bedrijf Mantis (zie <https://www.mantisbt.org>).

Voor mijn stage waren Unity en Visual Studio de belangrijkste programma's.

2.2 Belangrijke termen en concepten

Vooraleer ik in het meer technische deel van mijn stage duik, geef ik graag even enkele belangrijke termen mee die gerelateerd zijn aan het spel 'Flotsam' en aan de game-industrie.

2.2.1 Townheart

De Townheart is het centrum van de stad. De speler start het spel met deze boot. Hieraan kan de speler gebouwen bouwen. De Townheart heeft ook opslagruimte voor materialen, water en energy. Via de kaart kan de speler de Townheart verplaatsen in de wereld en op zoek gaan naar nieuwe grondstoffen om te verzamelen.

2.2.2 'Landmark'

Een 'landmark' is een restant van de wereld voor de overspoeling. Dit kan een klein eiland zijn dat nog juist boven het zeeniveau uitsteekt, maar dit kan eveneens een kerk of appartementsgebouw zijn. Verschillende 'landmarks' bevatten verschillende materialen die de spelers kunnen verzamelen.

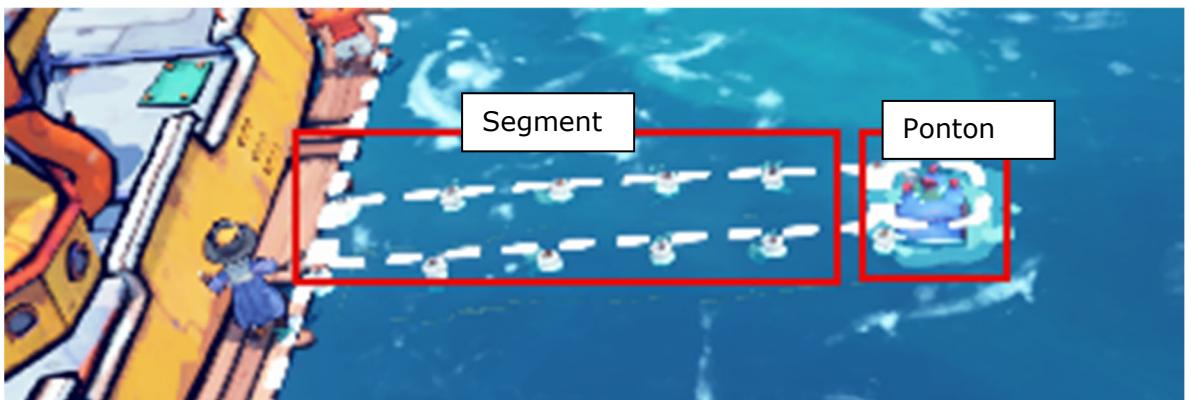
¹ Version control is een systeem dat het beheer en de tracking van wijzigingen in bestanden en broncode mogelijk maakt, waardoor meerdere mensen efficiënt kunnen samenwerken aan een project.

2.2.3 'Drifter'

Een 'Drifter' is een personage in Flotsam. Een 'Drifter' is ofwel een inwoner van de stad van de speler ofwel is het een personage dat zich op een 'landmark' bevindt. Als een 'Drifter' zich op een 'landmark' bevindt, kan de speler die 'Drifter' redden en zal de 'Drifter' mee inwoner worden van zijn stad.

2.2.4 'Walkway'

Een 'walkway' is een speciaal soort gebouw dat uit 2 delen bestaat: een segment en een ponton. Het doel van 'walkways' is om verschillende delen van de stad met elkaar en de Townheart te verbinden. De speler kan gebouwen bouwen aan een 'walkway'. Hierdoor is hij niet meer beperkt tot het enkel kunnen bouwen aan de Townheart.



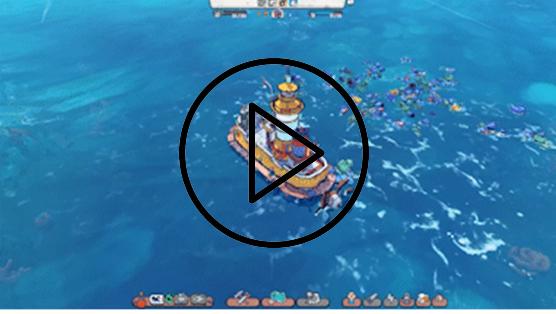
Afbeelding 1: Een 'walkway' met een segment en een ponton.

Een 'walkway' segment kan je zien als een plank die je ergens tussen legt. Terwijl normale gebouwen telkens maar aan één punt moeten vasthangen, moet een segment telkens aan 2 punten vasthangen, een startpunt en een eindpunt.

Een 'walkway' ponton zorgt ervoor dat een segment altijd een eindpunt kan hebben. In veel gevallen, zoals in afbeelding 1 hierboven, wilt de speler een 'walkway' bouwen weg van de Townheart, zodat hij de stad kan uitbreiden. Als de speler dit probeert te doen, zal dit in de volgende stappen gebeuren:

1. De speler selecteert de 'walkway' in het bouwmenu.
2. De speler plaatst de preview aan een valide punt. Dit kan de Townheart zijn, maar even goed een andere 'walkway'.
3. De preview van de 'walkway' zet zich nu vast op het punt waar de speler heeft geklikt en de speler krijgt een uitgebreidere preview. Het punt waar de speler heeft geklikt is het startpunt van de 'walkway'.
4. De speler kiest nu een plaats om het eindpunt van de 'walkway' te plaatsen. Klikt de speler op een andere 'walkway', dan wordt die 'walkway' het eindpunt. Als de speler het eindpunt in het midden van de zee wil leggen, dan wordt er een ponton gecreëerd.

De 'drifters' gebruiken de 'walkways' ook om zich te verplaatsen binnen de stad.

<p>De speler bouwt een 'walkway' aan de Townheart. Het startpunt van de 'walkway' is de Townheart. Het eindpunt ligt in de zee, dus er wordt een ponton bijgebouwd om als eindpunt te dienen.</p>	<p>De speler bouwt een 'walkway' tussen twee andere 'walkways'. Zowel het begin als het eindpunt zijn in dit geval dus 'walkway' segmenten. Er wordt in dit geval geen extra ponton gebouwd, omdat het eindpunt al bestaat.</p>
	

Afbeelding 2: Het verschil tussen een 'walkway' bouwen met ponton (links) en zonder ponton (rechts).

2.2.5 'Overlay'

Een 'overlay' is simpelweg een visueel effect, dat de speler laat weten dat hij in een bepaalde mode zit. Zo bestonden er bij de start van mijn stage al 2 'overlays': de 'beauty overlay' en de 'energy overlay'.

Als de 'beauty overlay' actief is, verschijnt er een groen kader rond het scherm en ziet de speler de schoonheidsscore van elk gebouw boven het gebouw zweven.

Als de 'energy overlay' actief is, verschijnt er een geel kader rond het scherm en ziet de speler extra informatie in verband met het energiesysteem van het spel.

Later in mijn stage is er nog een 3^{de} 'overlay' toegevoegd, namelijk de 'weight overlay'. Als die actief is, verschijnt er een grijs kader en ziet de speler het gewicht van alle geplaatste gebouwen.

'Overlays' zijn in principe een detail in het spel. De rede waarom ik het hier toch vermeld, is omdat ik in één van mijn latere aanpassingen gebruik maak van een 'overlay' om het voor de speler visueel duidelijk te maken dat hij in een speciale modus zit binnen in het spel. Meer hierover in hoofdstuk 2.3, de 'Architect overlay'.

2.2.6 Upgraden van gebouwen

De bouw van 'walkways' en de bouw van andere gebouwen maakt integraal deel uit van het spel. De manier waarop de speler vooruitgang maakt, is dan ook door het bouwen van specifieke gebouwen die de noden en vragen van de drifters vervullen.

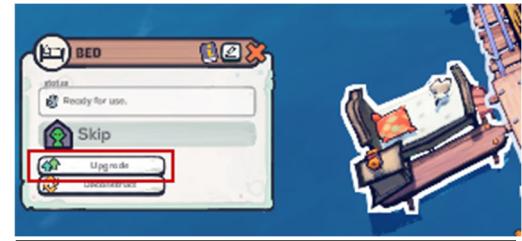
Een belangrijk concept hier is dat elk gebouw moet vasthangen aan de Townheart (de boot waarmee de speler start). De speler kan een gebouw bouwen op 2 manieren: 1) rechtstreeks aan de Townheart of 2) verbonden aan walkways, die op hun beurt weer verbonden zijn met de Townheart.



Afbeelding 3a: Vasthangend gebouw

Een speler kan bepaalde gebouwen upgraden. Dit doet hij door op het gebouw te klikken. Op dat moment verschijnt er een infopaneel. Indien het gebouw een geüpgraded versie heeft, zal de speler op het infopaneel een upgrade-knop zien. Pas als de speler aan alle nodige vereisten voldoet, zal de knop interactief zijn. De vereisten zijn meestal de volgende:

- De speler moet de relevante technologie hebben onderzocht.
- De speler moet de nodige grondstoffen hebben voor de upgrade.

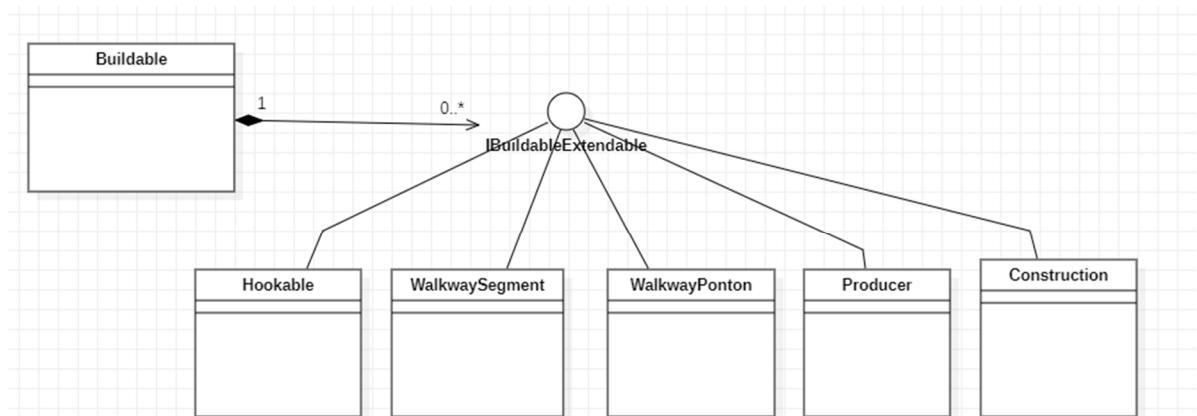


Afbeelding 3b: Upgrade knop

De upgrade van een gebouw gebeurt technisch in 2 stappen:

1. Het bestaande gebouw wordt volledig verwijderd.
2. Het geüpgraderde gebouw wordt op dezelfde plaats neergezet (met dezelfde instellingen).

De instellingen verschillen van gebouw tot gebouw, maar zijn in de code geïmplementeerd door middel van verschillende IBuildableExtendable klassen, die het Decorator patroon volgen (zie afbeelding 4).



Afbeelding 4: UML Klassendiagram van de Buildable klasse met enkele voorbeelden van IBuildableExtendable componenten.

2.2.7 ‘CursorManager’ en ‘CursorProperties’ klasse

‘Cursorproperties’ bepalen het gedrag dat de muiscursor heeft op verschillende momenten in het spel. Dit gaat bijvoorbeeld over wat er moet gebeuren wanneer de speler op de linkermuisknop drukt of zelfs wat er elke frame moet gebeuren terwijl de ‘CursorProperty’ actief is.

Twee voorbeelden van ‘CursorProperties’ zijn:

- ‘SingleConstructionCursorProperties’
- ‘SalvageMarkerCursorProperties’

De ‘SingleConstructionCursorProperties’ wordt geactiveerd wanneer de speler in de ‘Buildmenu’ op een gebouw klikt. Dit zorgt ervoor dat de cursor andere functies krijgt. Eerst en vooral ziet de speler nu dat de muis gevuld wordt door een transparante versie van het gebouw dat hij heeft geselecteerd. Die transparante versie wordt een ‘preview’ genoemd en kleurt groen als de speler het gebouw kan plaatsen en kleurt

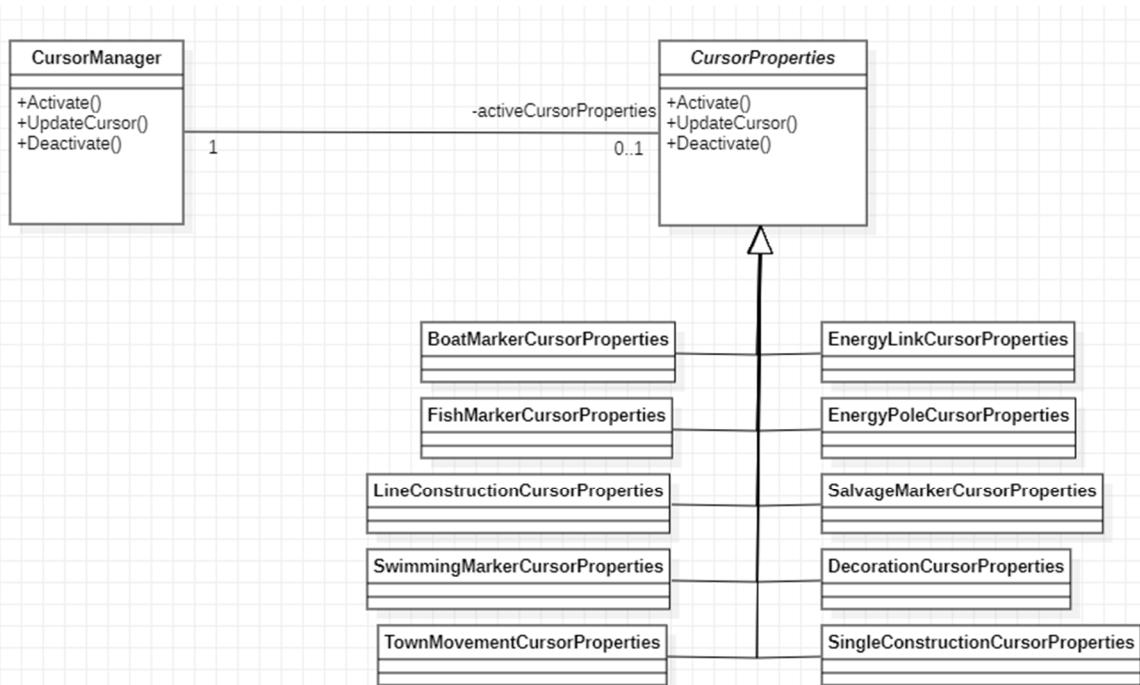
rood als de speler het gebouw niet kan plaatsen. Daarnaast verandert ook de functie van de linker- en rechtermuisknop. Als de speler op de linkermuisknop drukt, zal het gebouw, indien mogelijk, geplaatst worden. Als de speler op de rechtermuisknop drukt, zal de 'SingleConstructionCursorProperties' teruggedactiveerd worden.

De 'SalvageMarkerCursorProperties' verandert de functies van de muiscursor op een andere manier. Het zorgt voor een preview van een boei. De speler kan deze boei plaatsen in de zee in de buurt van materialen door op de linkermuisknop te drukken.

In Flotsam zijn er nog een heel aantal verschillende 'CursorProperties', die allemaal hun eigen functies hebben en in verschillende situaties geactiveerd zullen worden.

Bekijken vanuit een designpatroon oogpunt, zijn 'CursorProperties' implementaties van het 'Strategy' patroon.

Het spel heeft nog een andere klasse, de 'CursorManager' genoemd. Dit is een 'Singleton', die de verschillende 'CursorProperties' activeert, deactiveert en managet. Als de speler op de knop voor de 'SingleConstructionCursorProperties' klikt, wordt de activeringsmethode opgeroepen bij de 'CursorManager'. De 'CursorManager' houdt ook bij welke 'CursorProperty' er geactiveerd is en roept elke frame de 'UpdateCursor' methode aan van de geactiveerde 'CursorProperties'. Dit bepaalt het gedrag van de muiscursor op dat moment. (afbeelding 5)



Afbeelding 5: UML klassesdiagram van de CursorManager met CursorProperties

2.2.8 'EnergyPoles'

'EnergyPoles' zijn speciale gebouwen die op een 'walkway' ponton kunnen gebouwd worden. Ze zorgen ervoor dat de speler gebouwen kan verbinden op een 'energy grid', zodat gebouwen die energie nodig hebben, kunnen worden verbonden met gebouwen die energie produceren en gebouwen die energie opslaan. (afbeelding 6)

Op afbeelding 6 zijn er 2 'energy poles' te zien. Ze zijn telkens op een 'walkway' ponton gebouwd. Vanuit elke 'energy pole' lopen er één of meerdere kabels naar andere gebouwen op het 'energy grid'.



Afbeelding 6: Twee 'energy poles' verbonden met elkaar en verbonden met enkele andere gebouwen.

3 PROJECTFASES

Tijdens mijn stage kwam ik een reeks aan problemen tegen. Enkele bugs van het spel waren reeds gekend voor de aanvang van mijn stage. Andere problemen kwam ik pas tegen tijdens mijn stage. Door 1 probleem op te lossen, komt namelijk al snel een 2^{de} tevoorschijn. Samen met mijn mentor heb ik alle hindernissen besproken en 1 voor 1 onderzocht.

De problemen die zich voordeden, kan ik opdelen in 3 grote groepen:

- 1) Het oplossen van bestaande, gekende bugs.
- 2) Het verbeteren van de 'Walkways'.
- 3) Het optimaliseren van de 'Architect Overlay'.

Stap voor stap leg ik je uit welke stappen ik tijdens mijn stage heb doorlopen om de gekende en nog niet gevonden problemen op te lossen. Per problemengroep geef ik je telkens eerst algemeen een uitleg van de problemen die zich voordeden en wat ik heb geleerd in iedere fase. Hierna geef ik je enkele voorbeelden en hoe ik deze problemen heb opgelost.

3.1 Bestaande bugs en problemen in het spel

Eén van de belangrijkste zaken die mij is bijgebleven uit deze fase is dat bij het oplossen van bugs het altijd belangrijk is om te zoeken naar de oorzaak. Dit klink soms voor de hand liggend, maar een bug is soms gemakkelijk op te lossen door snel een uitzondering in te bouwen. Op korte termijn is dit een goede en betrouwbare oplossing. Op lange termijn zorgt dit echter voor complexiteit en andere problemen.

Daarnaast heeft deze fase mij ook geleerd hoe belangrijk documenteren is. Iedere bug en feature die ik tegenkwam, gaf ik een beschrijving, een stappenplan voor reproductie en eventuele opmerkingen of oplossingen. Hiervoor gebruikte ik een sjabloon (afbeelding 7). Door iedere bug, probleem en verloop hoe de bug op te lossen, te noteren, verkreeg ik een goed overzicht. Zeker wanneer ik op een gegeven moment 5 bugs achter elkaar vond, heeft me dit goed geholpen.

Ik ben gestart met het oplossen van de bestaande bugs en problemen in het bouwsysteem. Dit gaf me de mogelijkheid om grondig kennis te maken met de codebase en het spel in het algemeen.

Hieronder zal ik een aantal bugs toelichten, met daarbij telkens beeldmateriaal van de situatie voor en na de geïmplementeerde oplossing. Het beeldmateriaal bestaat uit geanimeerde afbeeldingen van het spel.

3.1.1 'Difter' zit vast op 'walkway'

Dit was een kleine bug. Er loopt iets mis bij het afbreken van de 'walkway'. De 'difter' blijft vastzitten op de 'walkway', en hij wordt niet naar zijn Townheart teruggeplaatst. Hierdoor wordt de 'walkway' niet verwijderd wanneer dit zou moeten. De grote oorzaak van het probleem ligt bij de navigeren van de 'drifter'.

Description

When deconstructing a Walkway, a drifter might idle on a connected Walkway Segment. When this happens and another drifter comes to pick up the resources, the game is unable to remove the Walkway Segment. The drifter gets stuck and the Walkway stays in the scene with the greyed out deconstruction icon above.

The following errors get thrown the moment the second drifter picks up the resources:

```
[11:33:00] NullReferenceException: Object reference not set to an instance of an object
    Navigator.UpdateGraph (PathfindingNode node) (at Assets/Code/Pathfinding/Navigators/Navigator.cs:700)
[11:33:08] NullReferenceException: Object reference not set to an instance of an object
    PathfindingNode.Target_Context (PathfindingNode node) (at Assets/Code/Pathfinding/Target/PathfindingNodeTarget.cs:26)
```

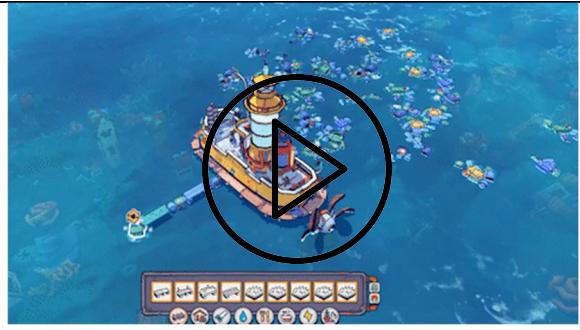
Steps to reproduce

1. Construct a Walkway
2. Deconstruct the Walkway (preferably while there are no other pending tasks)

Afbeelding 7: Voorbeeld sjabloon voor het documenteren van bugs

Opmerkelijk bij deze bug is dat ik het heb kunnen oplossen met slechts één lijn aan code, maar zoals hierboven vermeld, ligt de moeilijkheid om een probleem op te lossen vaker bij het achterhalen van de echte oorzaak van het probleem, dan bij het implementeren van de oplossing zelf.

In de onderstaande afbeelding (afbeelding 8) kan je de situatie vergelijken, voor en na de aanpassingen. Door een verandering in de bouwstatus van de 'walkway', heb ik het probleem uiteindelijk kunnen oplossen.

Voor: De 'drifter' blijft op de 'walkway' staan en zit daarna vast. Hij kan niet meer bewegen en de 'walkway' wordt ook niet verwijderd.	Na: De 'drifters' worden correct teruggeplaatst naar de Townheart en de 'walkway' wordt verwijderd.
	

Afbeelding 8: 'Drifter' zit vast op de 'walkway'

3.1.2 Bouwen op 'walkways' in opbouw, geeft problemen

Gebruiksvriendelijkheid wordt door een speler zeer gewaardeerd. En speler verwacht tijdens het spelen van een spel dat het bouwen van boten en gebouwen vlot en gemakkelijk verloopt. Is dit niet zo, dan speelt de gebruiker liever een ander spel.

Deze aanpassing was vooral één van gebruiksvriendelijkheid. Voor de gemaakte aanpassing kon een gebouw in sommige gevallen niet geplaatst worden aan een 'walkway', die nog in opbouw was. Dit had te maken met de positie van bepaalde hoekpunten op de navigatie graaf die niet dicht genoeg bij elkaar waren om zich met elkaar te kunnen verbinden.

Nu, na de aanpassing, kan een speler zijn stad uitbreiden, zonder telkens te moeten wachten tot dat elke 'walkway' gebouwd is. (afbeelding 9)

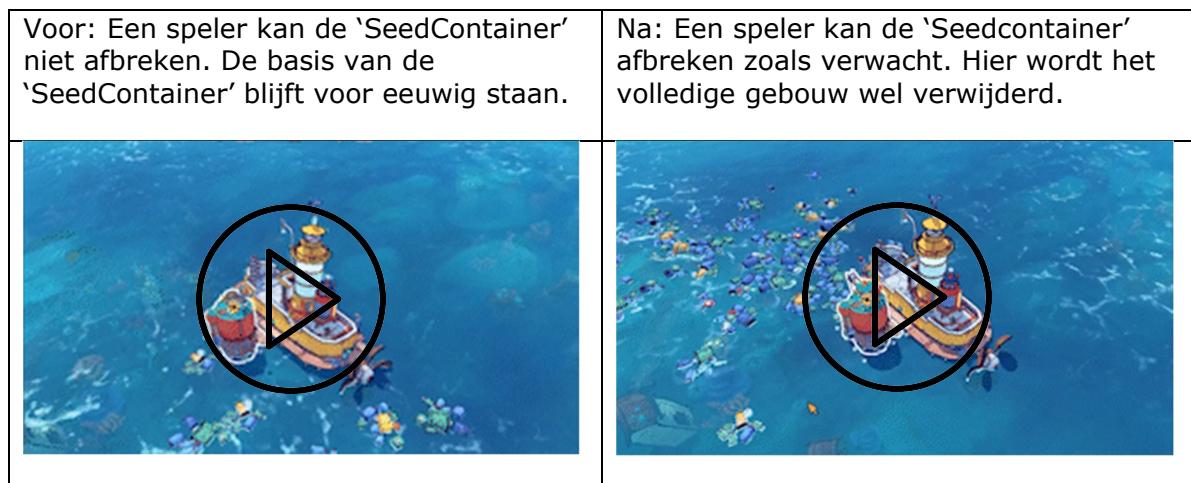
Voor: Gebouwen kunnen op schijnbaar willekeurige plaatsen niet geplaatst worden.	Na: Gebouwen kunnen gewoon worden geplaatst zoals de speler zou verwachten.
	

Afbeelding 9: Bouwen op 'walkways' in opbouw

3.1.3 'SeedContainer' kan niet afgebroken worden

Een 'Seedcontainer' is een specifiek gebouw in het spel. Het is een opslagplaats voor zaadjes, waarmee de speler later bomen en bloemen kan planten.

Het gebouw heeft een inventaris. Normaalgezien heeft hij een inventaris voor één planten én zaden, maar het probleem was dat de inventaris enkel zaadjes aanvaardde. Dus wanneer een speler het gebouw afbreekt, kunnen de grondstoffen van het gebouw in geen enkele inventaris geplaatst worden. Hierdoor kan de speler het gebouw niet afbreken. Na mijn aanpassingen kon een speler de 'Seedcontainer' afbreken zoals verwacht. (afbeelding 10)



Afbeelding 10: Afbreken van de 'Seedcontainer'

3.1.4 Een andere drifter vervoert de materialen

Het is de bedoeling dat wanneer een drifter een gebouw afbreekt, hij de materialen meeneemt naar de dichterbijzijnde opslagplaats. In dit geval is dit de Townheart (zie afbeelding 11), maar dit was niet het geval. Ik merkte op dat een andere drifter dan degene die het gebouw had afgebroken, de materialen ging ophalen en vervoeren.

Om dit probleem op te lossen moesten er 2 aanpassingen gemaakt worden. De eerste aanpassing was aan de startpositie van de vervoerstaak. Bij het kiezen van een 'drifter' werd er gekeken naar welke 'drifter' zich het dichts bij de opdracht bevindt. Bij een taak zoals bouwen of afbreken ging dit goed, maar bij de taak 'materiaal ophalen', werd de afstand berekend tussen de 'drifter' en de opslagplaats. Maar in zo'n vervoersproject moet de 'drifter' eerst naar de materialen lopen en daarna pas naar de opslagplaats.

Nadat ik dit had aangepast, merkte ik dat het nog steeds niet werkte. Dit kwam omdat de 'drifter' die het afbreken juist had uitgevoerd nog niet beschikbaar was voor de volgende taak. Dit heb ik opgelost door het aanmaken van de vervoerstaak een 'frame' uit te stellen. Op die manier waren alle 'drifters' terug beschikbaar.

Mijn aanpassingen waren geslaagd. Wanneer ik nu een opdracht geef om een gebouw af te breken, is het ook deze drifter, die het dichtste bij de afbraak staat, die de grondstoffen ophaalt en wegbrengt. (afbeelding 11).

Voor: Niet de 'drifter' die het gebouw heeft afgebroken, maar een andere 'drifter', die op de boot zit, komt de grondstoffen ophalen.	Na: De 'drifter' die het gebouw heeft afgebroken, brengt eveneens de grondstoffen weg, naar de boot.
	

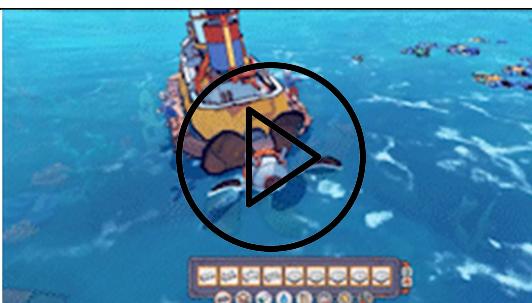
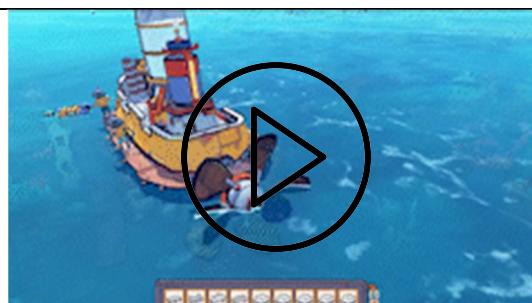
Afbeelding 11: De dichtstbijzijnde 'drifter' haalt de materialen op.

3.1.5 Blokkeringsregio van de Townheart motor

De motor achteraan de Townheart blokkeert de bouw van nieuwe gebouwen. De motor voorkomt vooral dat de speler orthogonaal kan bouwen. Daarbij is de geblokkeerde regio niet volledig symmetrisch, wat voor meeste spelers, die graag symmetrisch bouwen, zeer vervelend aanvoelt. (afbeelding 12)

Rond de motor bevindt zich in het spel een onzichtbare polygoon. Deze bepaalt of er in die regio gebouwd kan worden of niet. Om het bovenstaande probleem op te lossen, heb ik de grootte van die polygoon aangepast. Om alles er consequent te doen uitzien, heb ik daarna ook de grootte van de motor en van de rode regio aangepast.

Ook deze aanpassing is een 'Quality of life'-aanpassing. Ook al is het een kleine verandering, ook een kleine frustratie hoort niet thuis in een spel waar de speler zich juist moet kunnen amuseren.

Voor: De speler kan niet bouwen rondom de motor en de bouw is asymmetrisch.	Na: De speler kan bouwen rondom de motor. Het bouwen voelt toegefeliiger aan en de bouw is symmetrisch.
	

Afbeelding 12: Blokkeringsregio van de Townheart

3.1.6 Het afbreken van 'walkway' segmenten

Zoals hierboven al aangehaald tijdens de uitleg over de 'walkways', bestaan de 'walkways' uit 2 delen: een segment en een ponton. Segmenten zijn de grootste en duidelijkste delen van een 'walkway'. Ze zijn veel groter dan de pontons, maar om een 'walkway' te kunnen afbreken, moest de speler tot nu toe altijd op de ponton klikken. Dit was niet alleen niet duidelijk, maar voelde als speler ook zeer vreemd aan.

Er was slechts 1 uitzondering. Wanneer een segment aanwezig was tussen 2 andere segmenten dan kon de speler op het middelste segment klikken om de brug te verwijderen.....

Met mijn aanpassingen kan een speler nu zowel op het segment als op de ponton klikken om een 'walkway' af te breken. De afbraak voelt nu veel intuïtiever aan. (afbeelding 13) De oplossing vond ik door het uitvoeren van aanpassingen in de code van de segmenten en van de pontons.

Voor: De speler kan het segment niet afbreken. Enkel wanneer hij op de aanliggende ponton klikt.	Na: De speler kan ook via het segment de 'walkway' afbreken. Wanneer de speler op een segment klikt, verschijnt hier ook de 'afbreken'-knop. Net als bij het afbreken via de ponton.
	

Afbeelding 13: Het afbreken van een 'walkway'-segment

Zoals hier te zien is, is de bepalende factor voor het afbreken van de 'walkway', de ponton. Dit betekent dat de materialen in de ponton zitten na de afbraak van de 'walkway'. De segmenten dragen bij aan de 'Beauty'-score van de stad. Aangezien dat segmenten geen materialen kosten, kan de speler de 'walkways' gratis bouwen door een segment tussen 2 andere segmenten te plaatsen. Dit was zeer onlogisch en heb ik in een latere fase ook nog aangepast.

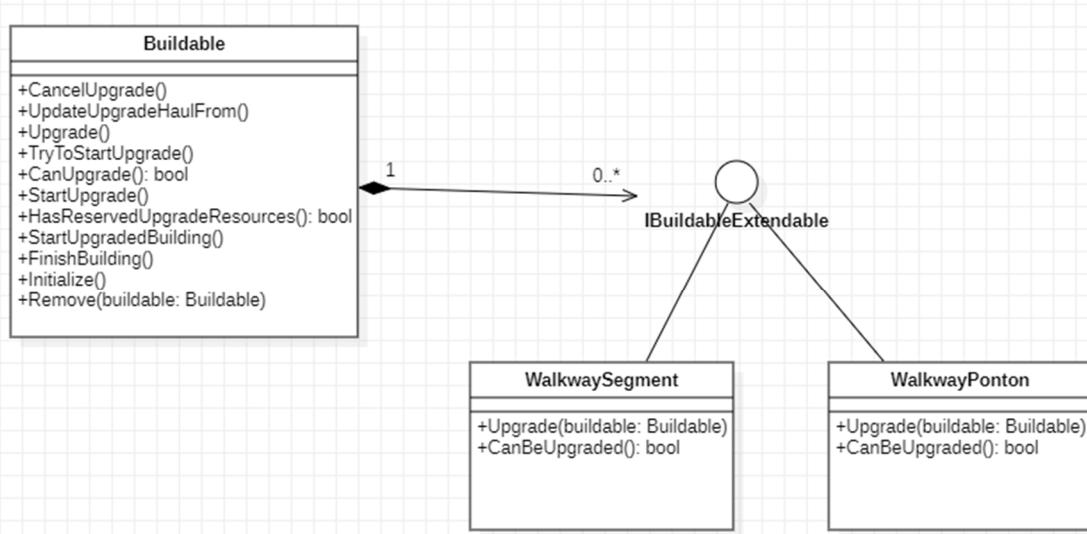
3.2 Upgraden van 'walkways'

Na het oplossen van een aantal van de bestaande bugs werd er van mij verwacht om mijn eigen features te beginnen toevoegen. De eerste grote toevoeging was het upgraden van de 'walkways'. Dit was een feature die de spelers al in het spel wilden hebben, maar waar het bedrijf nog geen tijd voor had kunnen vrijmaken.

3.2.1 'Buildable'-, 'WalkwaySegment'- en 'WalkwayPonton'-klasse

Het implementeren van de upgrade functionaliteit hield vooral in dat veel bestaande klassen moesten worden aangepast. De klassen die hier van belang zijn, zijn de Buildable-klasse, de WalkwaySegment-klasse en de WalkwayPonton-klasse. De manier waarop deze klassen aan elkaar gelinkt zijn, wordt weergegeven in afbeelding 14. Hier is ook duidelijk het 'Decorator'-patroon in te herkennen.

De 'Buildable'-, 'WalkwaySegment'- en 'WalkwayPonton'-klasse hebben een overvloed aan verschillende attributen en methodes. Degene die voor deze aanpassing van toepassing zijn, kan je eveneens terugvinden in afbeelding 14.



Afbeelding 14: Vereenvoudigd UML klassendiagram van de Buildable-, WalkwaySegment- en WalkwayPonton klasses.

3.2.2 'BuildableProperties'-bestand in de upgrade van walkways

Voor elk gebouw in het spel wordt er in Unity een 'BuildableProperties'-bestand aangemaakt. Dit bestand bevat alle vaste gegevens die het spel nodig heeft voor een bepaald gebouw. Eén van die gegevens is wat de geüpgraderde versie van het gebouw is. Tot nu toe hadden 'walkways' nog geen upgrades. De eerste stap was dus het aanpassen van deze instelling.

3.2.3 De upgrade-methode van de WalkwaySegment- en WalkwayPonton-klassen

De volgende stap was om enkele specifieke aanpassingen te maken aan de 'Upgrade'-methode van de WalkwaySegment- en WalkwayPonton-klassen. Aan een 'walkway' kunnen namelijk andere gebouwen vasthangen. Wanneer een 'walkway' een upgrade krijgt, moeten de aanhangende gebouwen worden overgezet van de oude 'walkway' naar de geüpgraderde 'walkway'.

Dit gebeurt in 2 stappen:

- Stap 1, neem alle gebouwen die aan de oude 'walkway' hingen.
- Stap 2, verbind al deze gebouwen met de geüpgraderde 'walkway'.

De upgrade van 'walkways' gebeurt op dezelfde manier als de upgrade van gebouwen. Ook hier is de code geïmplementeerd door middel van verschillende **IBuildableExtendable** klasses, die het Decorator patroon volgen. Maar vermits een 'walkway' uit 2 verschillende gebouwen kan bestaan, namelijk uit een 'walkway' segment en een 'walkway' ponton, betekent dit dat het upgraden ook moet opsplitst worden in logica voor 2 verschillende gebouwen.

Op dit moment in het proces, kan de speler zijn 'walkway' segmenten verbeteren. Zo kan hij plastic wandelwegen omvormen in houten wandelwegen of grassige wandelwegen omvormen in stenen wandelwegen. Er bestaan 4 upgrades van wandelwegen:

- 'Plastic walkways' → 'Wooden walkways'
- 'Wooden walkways' → 'Tiled walkways'

- 'Tiled walkways' → 'Grassy walkways'
- 'Grassy walkways' → 'Tiled walkways'

Merk op dat de 'Tiled walkways' kunnen worden geüpgraded naar de 'Grassy walkways', maar ook andersom. Dit is omdat deze 2 'walkways' dezelfde bonusen geven en dezelfde kost hebben qua grondstoffen. Het verschil tussen de 2 is dus enkel esthetisch. Op deze manier kan de speler makkelijk kiezen welke stijl hij zelf het beste vindt passen bij de stad die hij wil bouwen.

3.2.4 Gratis 'walkways'

Tijdens het implementeren van de upgrade functionaliteit, werd er een ander probleem ontdekt. Er was namelijk nog een algemener aanpassing nodig aan de werking van de 'walkways', vooraleer de upgrades goed konden geïmplementeerd worden. De speler kon namelijk gratis 'walkway' segmenten bouwen tussen 2 andere segmenten. Zo kon hij zijn stad verbeteren zonder materialen te investeren. Deze mogelijkheden waren in tegenstrijd met het basisconcept van het spel.

Daarnaast kreeg een speler gratis Beauty-score door het bouwen van mooiere 'walkway' segmenten, bv. 'Tiles walkways' of 'Grassy walkways'. Als op dit moment de upgrade feature geïmplementeerd werd, krijgt een speler ook gratis Beauty-score iedere keer dat hij op de knop 'upgrade' drukt.

Daarom was het noodzakelijk om een aanpassing door te voeren voor het implementeren van de upgrade feature. Nu, na de aanpassing, kan een speler gratis 'walkway' pontons bouwen, maar heeft hij voor de bouw van de 'walkway' segmenten wel materialen nodig. De pontons dragen namelijk op geen enkele manier bij aan de waarde van de stad, dus het kan geen kwaad dat ze gratis zijn. Zoals hiervoor al aangehaald, bestaan de 'walkway' pontons enkel om een eindpunt te voorzien voor een 'walkway' segment.

Door de bovenstaande aanpassing moeten de drifters nu ook de nodige materialen naar een 'walkway' segment brengen voordat het kan worden gebouwd. Het enige verschil tussen de 'walkway' segmenten en andere gebouwen is dat de drifters een 'walkway' segment niet moeten construeren. Met andere woorden: na het aanleveren van de materialen wordt een 'walkway' segment direct naar Buildstate "Finished" gezet. Andere gebouwen krijgen eerst nog de Buildstate "Building".

3.3 'Architect overlay'

'Architect overlay' stelt de speler in staat om zijn stad te kunnen herinrichten. Een speler verplaatst reeds gebouwde gebouwen en plaats deze terug hoe hij dit zelf wil. De gebouwen moeten natuurlijk wel op een valide plaats gezet worden. Dit wil zeggen dat ook verplaatsde gebouwen telkens verbonden moeten zijn met de Townheart.

Tijdens het ontwikkelen van deze feature werden er 3 moeilijkheden ontdekt:

- De functionaliteit 'Architect Overlay' moet visueel duidelijk zijn.
- De speler moet gebouwen niet alleen kunnen verplaatsen, maar ook tijdelijk kunnen opslaan.
- De speler moet de opgeslagen gebouwen kunnen meenemen wanneer hij zijn spel opslaat, om dan een volgende keer te kunnen verder spelen

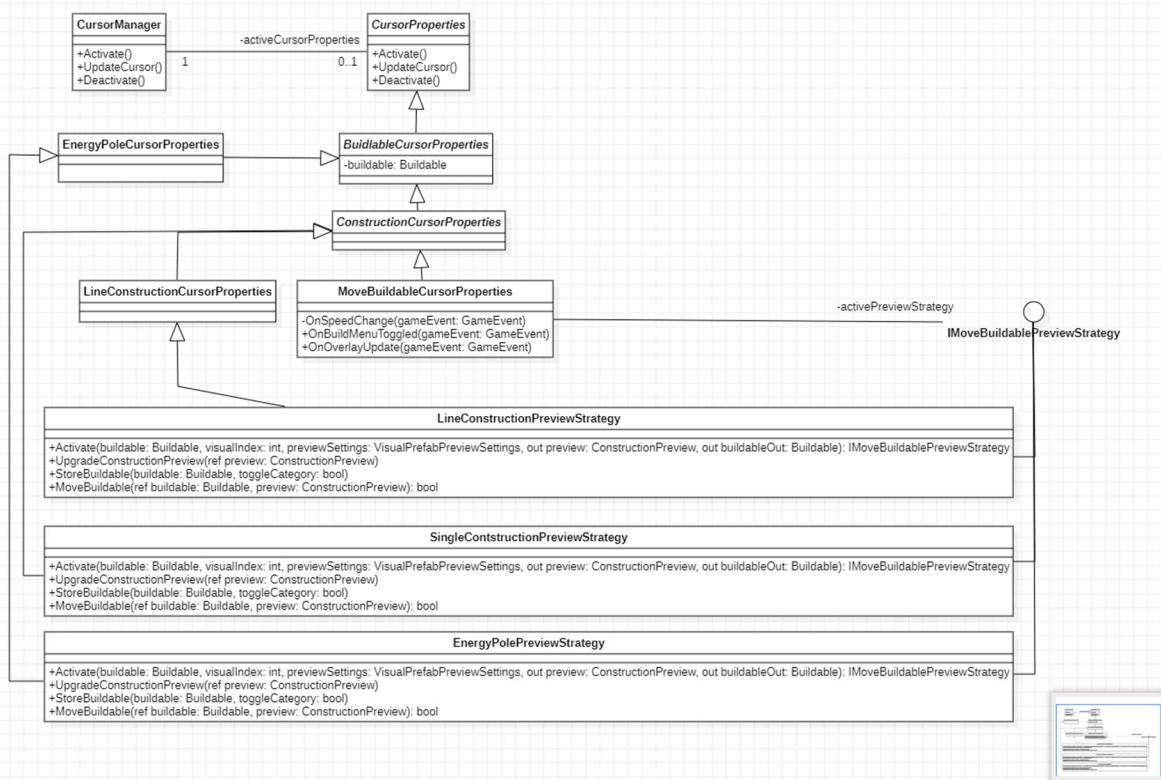
3.3.1 'Ups-and-downs' in het implementeren van de feature 'Architect Overlay'

Na het implementeren van de eerste echte feature (het upgraden van de 'walkways'), was de 'Architect overlay' het volgende grote project.

Dit project had wel enkele tegenslagen. Heel wat features zijn geïmplementeerd, maar werden later in het project weer helemaal aangepast of zelfs weggegooid. Het was veel 'trial and error' om een systeem te krijgen dat vlot en leuk aanvoelde voor de speler. De 3^{de} moeilijkheid in het ontwikkelen van de 'Architect Overlay', namelijk dat een speler zijn opgeslagen gebouwen kan meenemen tussen speelsessies door, is niet verwezenlijkt. Dit was een zeer complexe voorwaarde en naar onderhoudbaarheid toe, was het beter om dit niet te implementeren.

3.3.2 Het ontwikkelen van de 'Architect Overlay'

De uiteindelijke code voor de 'Architect Overlay' bevindt zich voor het grootste deel in nieuwe klassen. Veel van die klassen erven echter wel over van reeds bestaande klassen. Dit is een manier om zoveel mogelijk bestaande code te hergebruiken. In de volgende hoofdstukken wordt een gedetailleerde uitleg gegeven bij elke toegevoegde klasse. De samenhang tussen deze klassen kan in de onderstaande afbeelding worden bekijken. (afbeelding 15)



Afbeelding 15: UML klassendiagram van de MoveBuildableCursorProperties met gerelateerde klassen

3.3.3 De 'MoveBuildableCursorProperties' klasse

De 'MoveBuildableCursorProperties' is de belangrijkste klasse voor de 'Architect overlay'. Het bevat alle logica om te bepalen hoe een speler een gebouw kan selecteren, oppakken, opslaan en verplaatsen. Deze klasse is een uitbreiding van de

CursorProperties klasse en kan dus, zoals alle andere CursorProperties, geactiveerd worden via de CursorManager.

In dit geval, wordt de 'MoveBuildableCursorProperties' geactiveerd als de speler op de 'Architect overlay'-knop drukt. Dit is een nieuwe knop die toegevoegd is aan de onderkant van het scherm. Deze knop roept een methode aan in de CursorManager en geeft hierbij de 'MoveBuildableCursorProperties' mee.

Vooraleer de activatie van de 'MoveBuildableCursorProperties' goed kon verlopen, was het nodig om een aantal andere zaken te activeren, herprogrammeren en aan te passen. De meeste van deze nodige acties bestonden reeds in het spel (recht en niet onderlijnd). Andere zijn door mij nog geprogrammeerd (schuingedrukt en onderlijnd).

- Het activeren van een nieuwe visuele 'overlay'
- Het activeren van de 'construction border'
- Het juist zetten van de correcte 'UIState'
- Het spel moet gepauzeerd worden
- Het activeren van de 'ArchitectBuildMenu'
- Het toevoegen van een aantal 'EventListeners'

'EventListeners' zijn nodig opdat een spel zich in alle situaties op een correcte manier kan hervatten. Als de speler terug op de 'Architect overlay'-knop drukt, zal de 'Architect' modus zich afsluiten en zal het spel zich hervatten. Maar hetzelfde moet gebeuren als de speler het spel uit pauze haalt, of als de speler probeert de gewone bouwmenu te openen. De 'EventListeners' zorgen ervoor dat de 'Architect' modus zich ook in die gevallen correct afsluit.

Het 'EventListener' systeem is standaard in Unity aanwezig is. Andere programmeurs in mijn stagebedrijf hebben hier in het verleden reeds uitbreidingen op geschreven en ik heb hier dan ook gebruik van heb gemaakt. De volgende 3 listeners zijn door mij nog in de 'MoveBuildableCursorProperties' geïmplementeerd. Deze werken intern volgens het Observer pattern:

```
GameEventDispatcher.AddListener(GameEventType.GameSpeedChange, OnSpeedChanged);
GameEventDispatcher.AddListener(GameEventType.OverlayUpdate, OnOverlayUpdate);
GameEventDispatcher.AddListener(GameEventType.BuildableMenuToggled, OnBuildMenuToggled);
```

Afbeelding 16: Broncode toevoegen van 'EventListeners' in de 'MoveBuildableCursorProperties' klasse.

De bovenstaande code, voegt per lijn een 'EventListener' toe. Een 'EventListener' zal een meegegeven functie uitvoeren als er op een andere plaats een dispatch methode wordt aangeroepen voor die bepaalde eventtype. Kort samengevat zullen de 3 bovenstaande methodes allemaal de 'MoveBuildableCursorProperties' deactiveren, maar op een lichtjes andere manier, maar telkens rekening houdend met de manier waarop de speler de deactivatie heeft aangeroepen.

3.3.3.1 'IMoveBuildablePreviewStrategy' klasse

Als start voor de 'MoveBuildableCursorProperties'-klasse te programmeren werd de 'SingleConstructionCursorProperties'-klasse gebruikt. Vermits deze 'CursorProperties' verantwoordelijk is voor het bouwgedrag van de meeste gebouwen in het spel, was dit een goede basis. De speler verwacht bij het verplaatsen van gebouwen namelijk telkens hetzelfde gedrag.

Verder in de ontwikkeling werden er echter wel gebouwen gevonden die op een andere manier moesten verplaatst worden. Er bestaan namelijk andere 'CursorProperties' voor

die gebouwen: de 'LineConstructionCursorProperties' voor 'walkways' en de 'EnergyPoleConstructionCursorProperties' voor 'energypoles'. De beste oplossing, zodat deze ook in rekening werden gehouden, was het dynamisch aanpassen van de 'MoveBuildableCursorProperties'-klasse op basis van welk gebouw een speler op dat moment selecteert. Hiervoor wordt er gebruikt gemaakt van het Strategy patroon.

De 'IMoveBuildablePreviewStrategy'-interface is de belangrijkste klasse voor dit Strategy-patroon. De volgende 3 klassen zijn hier concrete implementaties van:

- 'SingleConstructionPreviewStrategy'
- 'LineConstructionPreviewStrategy'
- 'EnergyPolePreviewStrategy'

Elke van deze klassen is een uitbreiding van hun respectievelijke 'CursorProperties', waardoor ik heel wat logica voor het bouwen/verplaatsen van gebouwen niet opnieuw moest uitvinden, maar gewoon kon hergebruiken. Hierdoor is deze toepassing zeer onderhoudsvriendelijk en werd hetzelfde gedrag niet steeds herhaald voor ieder verschillend soort gebouw.

In de 'IMoveBuildablePreviewStrategy' zijn de volgende methodes gedefinieerd (afbeelding 17):

- De 'Activate' methode
- De 'UpdateConstructionPreview'
- De 'StoreBuildable' methode
- 'MoveBuildable' methode

```
9 references
IMoveBuildablePreviewStrategy Activate(Buildable buildable, int visualIndex,
    VisualPrefabPreviewSettings previewSettings,
    out ConstructionPreview preview, out Buildable buildableOut);
4 references
void UpdateConstructionPreview(ref ConstructionPreview preview);
4 references
void StoreBuildable(Buildable buildable, bool toggleCategory);
4 references
bool MoveBuildable(ref Buildable buildable, ConstructionPreview preview);
```

Afbeelding 17: Methodes voor IMoveBuildableStrategy interface

De 'Activate' methode: wanneer een speler een gebouw selecteert, wordt deze methode aangeroepen. De methode zal ervoor zorgen dat de 'Strategy' klaar is om gebruikt te worden voor dit specifieke gebouw. Vervolgens zal de methode de 'Strategy' klasse zelf teruggeven als 'return value', naar de 'MoveBuildableCursorProperties'. Dit om het opslaan van de 'Strategy' in de 'MoveBuildableCursorProperties' klasse te vergemakkelijken.

De 'UpdateConstructionPreview' methode: deze methode update de 'ConstructionPreview'. De 'ConstructionPreview' is een doorschijnende versie van het gebouw, maar zonder enige logica die normaal aan het gebouw gekoppeld is. De methode zorgt ervoor dat de preview de muis van de speler volgt, vervolgens naar een valide 'Hookable' springt en tot slot blauw kleurt als het gebouw geplaatst kan worden.

De 'StoreBuildable' methode: deze methode zorgt ervoor dat het geselecteerde gebouw kan worden opgeslagen in de 'ArchitectBuildMenu'. Zo kan een speler eerst meerdere gebouwen opslaan, die hij vervolgens één voor één kan selecteren en plaatsen.

De 'MoveBuildable' methode: deze methode laat de speler toe om het geselecteerde gebouw te plaatsen. Zo verandert het gebouw van positie in de wereld.

Voor de rest bevatten deze klassen natuurlijk nog een aantal ander methodes en attributen om deze methodes te ondersteunen en gebruiken ze ook een aantal reeds bestaande methodes van hun 'parent' klassen.

3.3.4 StoredBuildables 'dictionary'

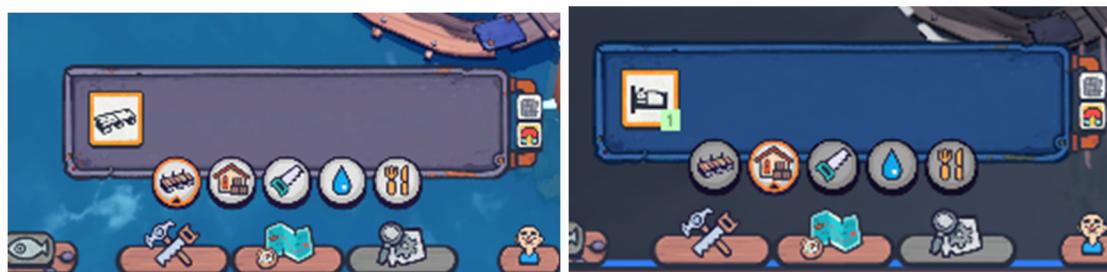
De gebouwen die een speler opslaat, worden bewaard in de 'dictionary' (verzamelplaats) van de 'Community' klasse. De 'Community' klasse is een reeds bestaande klasse, die onder andere bijhoudt welke objecten er allemaal aanwezig zijn in de stad van een speler. Dit leek me dus ook de meest logische plaats om de opgeslagen gebouwen op te slaan.

De 'StoredBuildables dictionary' is een lijst van 'Key/Value pairs' en neemt als 'key' (sleutel) de 'BuildableProperties' van het gebouw en als 'value' (waarde) een lijst van opgeslagen gebouwen van hetzelfde type. Elk soort gebouw in het spel heeft een unieke 'BuildableProperties'. Zo hebben alle gebouwen 'House' eenzelfde 'BuildableProperties' en alle gebouwen 'Small Storage' eenzelfde 'BuildableProperties', maar het gebouw 'House' heeft een andere 'BuildableProperties' dan het gebouw 'Small Storage'.

3.3.5 Het onderscheid van 'BuildableProperties' is belangrijk omdat dit ook bepaalt waar in de 'StoredBuildables dictionary' de gebouwen worden bewaard. De 'StoredBuildable dictionary' wordt op zijn beurt dan gebruikt om de gebouwen in de 'ArchitectBuildMenu' per soort te kunnen weergeven. Daarna kan de speler de 'StoredBuildables' opnieuw activeren door op de overeenkomstige knop in de 'ArchitectBuildMenu' te klikken. ArchitectBuildMenu

Het laatste puzzelstukje is de interface naar de speler toe. Zo goed als alle logica zit in de hierboven beschreven klasses. In die klasses wordt ook een deel van de interface logica gedaan, maar het grootste deel van de interface speelt zich af in de 'StoredBuildableCreation' klasse.

De 'StoredBuildableCreation' klasse is de klasse die verantwoordelijk is voor het maken en aanvullen van de 'ArchitectBuildMenu'. De 'ArchitectBuildMenu' is een uitbreiding van de normale 'BuildMenu', maar deze eerste is standaard leeg en zal aan de speler de knoppen laten zien voor alle 'StoredBuildables'.



Afbeelding 18: De normale 'BuildMenu' (links) en de 'ArchitectBuildMenu' (rechts)

In de normale 'BuildMenu' zijn alle gebouwen onderverdeeld in categorieën en worden er per category 'BuildableToggles' aangemaakt waar de speler op kan klikken als die genoeg materialen heeft om het gebouw te bouwen.

In de 'ArchitectBuildMenu', worden die 'BuildableToggles' ook allemaal aangemaakt, maar in dit geval maken we 'StoredBuildableToggles'. Deze zijn een extensie van de

normale 'BuildableToggles', maar passen het gedrag zo aan dat er een teller is op de knop die aangeeft hoeveel van dat soort gebouwen er beschikbaar zijn (zie afbeelding 18). Als er geen gebouwen beschikbaar zijn, zal de knop ook terug verdwijnen.

Een ander belangrijk verschil is dat de normale 'BuildableToggle', de corresponderende 'CursorProperties' voor het gebouw activeert, zodat de speler het geselecteerde gebouw kan bouwen. Een 'StoredBuildableToggle' roept telkens de reeds geactiveerde 'MoveBuildableCursorProperties' aan om het juiste gebouw uit de 'StoredBuildables dictionary' te gaan selecteren, zodat de speler het opgeslagen gebouw terug kan plaatsen.

3.3.6 Visuele 'overlay'

Wanneer een speler de 'MoveBuildableCursorProperties' activeert, activeert hij ook ineens de 'ArchitectOverlay' en de 'ArchitectBuildMenu'. De 'ArchitectOverlay' is een blauwe kader dat rond het scherm verschijnt om de speler duidelijk te maken dat hij in de 'Architect' modus bezig is.

4 CONCLUSIE

Het oorspronkelijke doel van mijn stage was om het constructie-systeem in Flotsam te verbeteren. Ik ben erin geslaagd om heel wat bugs op te lossen en enkele features toe te voegen. Op dat vlak ben ik ervan overtuigd dat mijn stage een succes was. Enkel van de 'Architect overlay', vind ik dat het niet volledig is afgeraakt. Ik heb alle logica geschreven en al heel wat gefinetuned, maar mijn stage is te vroeg afgelopen om de laatste puntjes op de 'i' te zetten en de feature in productie te zien. Over het algemeen ben ik echter wel tevreden met mijn andere aanpassingen en de 'walkway upgrade' feature die wel al in productie is gezet.

Als ik alles terug opnieuw zou kunnen doen, zijn er echter wel een aantal zaken die ik anders zou aanpakken. Ik had door gebrek aan communicatie aan mijn kant, te laat door wat precies de visie was van mijn bedrijf i.v.m. de 'Architect overlay'. Dit resulteerde in veel herschrijven en herdenken van code. Bij gelijkaardige projecten in de toekomst ben ik dus van plan om meer en sneller feedback te vragen van mijn opdrachtgever, zij dit een leidinggevende of een klant.

Voor de 'Architect overlay', zal mijn stagebedrijf zelf nog een paar zaken moeten afwerken. Er zullen nog correcte iconen moeten worden toegevoegd en ze zullen na nog wat testen, waarschijnlijk nog wat aanpassingen maken aan sommige functies van de 'Architect overlay'.

Algemeen ben ik heel blij dat ik mijn stage hier bij Pajama Llama Games heb aangelegd. Het was een heel andere omgeving dan de conservatieve business bedrijven, die ik gewoon ben. De soort logica die ik moest schrijven was ook anders en meer visuele. Er komt bij het maken van een game veel meer kijken dan enkel coderen en ik heb me hier besef dat ik als programmeur een deel uitmaak van een groter geheel.

Daarnaast heb ik op technisch vlak ook kennis gemaakt met complexere logica. In de game development wereld, ben ik veel meer mathematische logica tegengekomen. Om de posities van objecten in het spel te manipuleren, is een basiskennis lineaire algebra zeker een meerwaarde.

Ten slotte, ben ik tijdens mijn stage ook veel in aanraking gekomen met design patronen of opportuniteiten om design patronen te gebruiken. Dit is een ervaring die ik sowieso zal meenemen naar mijn volgend project.