Multiplatform Development

Skate L.A. - Documentatie

Door: Felix Buenen

Samenvatting

Skate L.A. is een mini skate game, gedeeltelijk gebaseerd op de game *Skate 3*. De speler legt een vooropgezet parcours af, waarin het de bedoeling is zoveel mogelijk punten te scoren. De speler kan deze punten scoren door trucs succesvol te landen. Skate L.A. is gemaakt voor twee platformen:

- Mac OS (bestuurd met een *PlayStation 4* controller)
- Android mobile

Concept

Ik vond het input systeem voor het spel *Skate 3* al langere tijd erg interessant: voornamelijk de responsive animatie van het character en de dynamische manier waarop trucs uitgevoerd kunnen worden (namelijk met de analoge stick). Na wat te brainstormen over mogelijke projecten zag ik een mogelijkheid om deze input te integreren voor zowel de gamepad als mobile input. Dit was dan ook het eerste uitgangspunt voor het skating game concept.

Uitwerking (platform dependence)

De platform dependence in Skate L.A. zit hem voornamelijk in de **UI** en **input**. Deze twee onderdelen zal ik nu verder toelichten.

UI

Ten eerste heeft het hoofdmenu op de Mac OS versie meer ruimte, omdat deze versie ook op een veel groter scherm gespeeld wordt dan de mobiele versie. Bij de mobiele versie ligt alles in het menu wat dichter bij elkaar, zodat de speler op het relatief kleine scherm snel op de juiste button kan klikken.

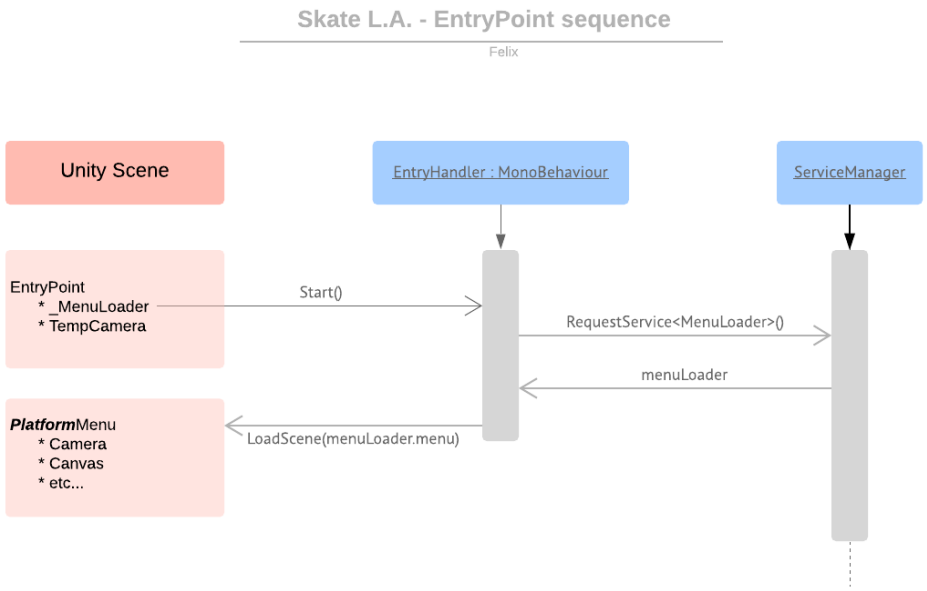
Verder heb ik ook een ‘controls’ menu toegevoegd. De speler kan hier zien wat de juiste beweeg patronen zijn voor specifieke trucs. Wederom speelt schermgrootte hier een rol: in de mobiele versie kan de speler door een scroll-panel swipen om alle trucs te zien. Bij de OS X versie staat alles op 1 scherm.

Android menu's

Mac OS menu's

Implementatie

De menu's heb ik in verschillende Unity scenes geïmplementeerd. Het enige wat Unity dan nog moet doen is de juiste scenes aanroepen op basis van de huidige build target. Unity heeft vóór het compilen een lijst nodig met de scenes die gecompiled moeten worden. Aanvankelijk probeerde ik een manier te vinden waarmee ik dynamisch scenes aan de compile-lijst kon toevoegen. Unity leek hier niet echt voor gebouwd, en onder het mom van "over-engineeren is ook niet goed" ben ik uiteindelijk met de volgende oplossing gekomen:



Unity laadt ten eerste de scene "EntryPoint" met alleen een Camera en een GameObject genaamd _MenuLoader. Deze heeft een EntryHandler. Dit object is verantwoordelijk voor het laden van het

juiste hoofdmenu. Dit doet hij door in de ServiceManager een SceneLoader op te vragen: de SceneLoader is een superklasse van PCSceneLoader en AndroidSceneLoader, welke verantwoordelijk zijn voor het onthouden van de namen van menu scenes voor Mac OS respectievelijk Android versie. De SerivceManager is een singleton die instanties van andere singletons op aanvraag weggeeft.

Nu de SceneLoader is gespecificeerd, kan op ieder moment in de game het juiste menu opgevraagd worden. Dit is handig voor wanneer de game voor het eerst start, maar ook wanneer de speler het controls scherm wil zien of bijvoorbeeld na het spelen van de game terug wilt naar het hoofdmenu.

In de hoofd scene *Main* (waar het spel zelf geïmplementeerd is) zit ook wat platform dependent code, namelijk wanneer de speler de optie krijgt om nog een keer te spelen of terug te gaan naar het hoofdmenu. Ik had dit op dezelfde manier kunnen doen als de menu handling. Echter, omdat het verschil zo minimaal is (het gaat om twee buttons die per platform verschillen) heb ik de conditional compilation in de game code zelf verwikkeld. In een groter project had ik hier een andere oplossing voor gekozen, bijvoorbeeld door het gebruik maken van de SceneLoader in combinatie met Unity's ingebouwde *additive scene loading*.

Code

`EntryHandler`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Misc/EntryHandler.cs
`SceneLoader`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Services/SceneLoader.cs
`PCSceneLoader`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Services/PCSceneLoader.cs
`AndroidSceneLoader`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Services/AndroidSceneLoader.cs
`ServiceManager`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Services/ServiceManager.cs

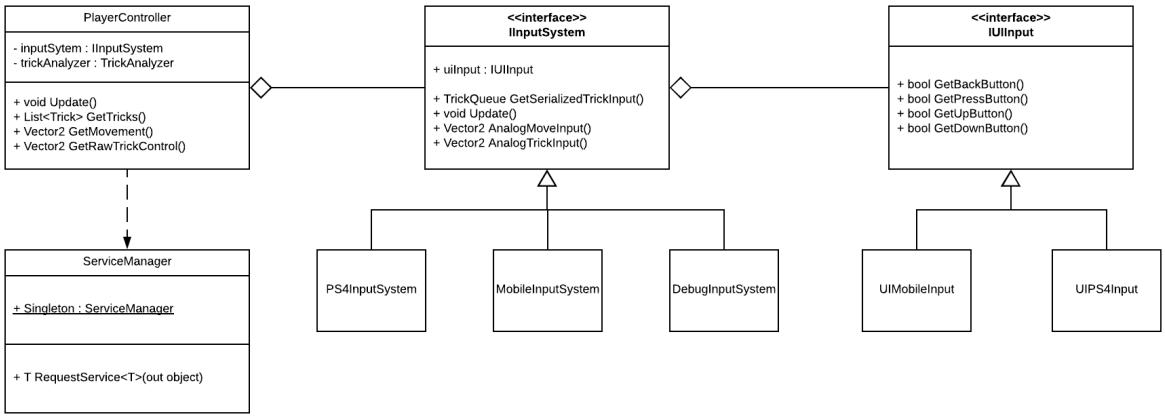
Input

De eerdergenoemde input van Skate 3 heb ik dus als startpunt genomen voor de PS4 input voor Skate L.A. Het porten naar mobile was een natuurlijke keuze: in plaats van een joystick te bewegen voor trucs, swipet de speler deze patronen gewoon direct op het scherm.

De skater gaat automatisch vooruit, maar moet wel bestuurd worden. Voor de Mac OS versie gebeurt dit met de linker joystick van de PS4 controller. Op de mobiele versie wordt de skater bestuurd door de mobiel te kantelen: dit zorgt ervoor dat de speler niet te maken heeft met on-screen buttons, en hij het spel op een intuïtieve manier kan spelen.

Tot slot is er natuurlijk nog UI-input. Voor de Mac OS versie heb ik onderscheid gemaakt tussen clickable buttons (in het menu, waar de speler met de D-Pad een button kan selecteren) en PS4-button clicks. Voor mobile heb ik alleen maar buttons gebruikt, omdat er geen sprake is van externe input zoals een PS4 controller.

Implementatie



In deze UML heb ik de PlayerController als enige gebruiker van IInputSystem weergegeven. In het project zijn er echter meerdere objecten die via ServiceManager de singleton instantie van IInputSystem opvragen en gebruiken. De ServiceManager is ook verantwoordelijk voor het initialiseren van een specifieke subklasse deze interface. Er is ook een IUIInput interface, die alleen gebruikt wordt voor UI input. Dit is een geneste klasse in IInputSystem. Omdat IUIInput redelijk klein is, leek me dit wel compact en ook handig voor initialisatie (de derived klassen van IInputSystem maken zelf hun instantie van IUIInput, omdat ze weten welke specifieke IUInput ze nodig hebben).

Code

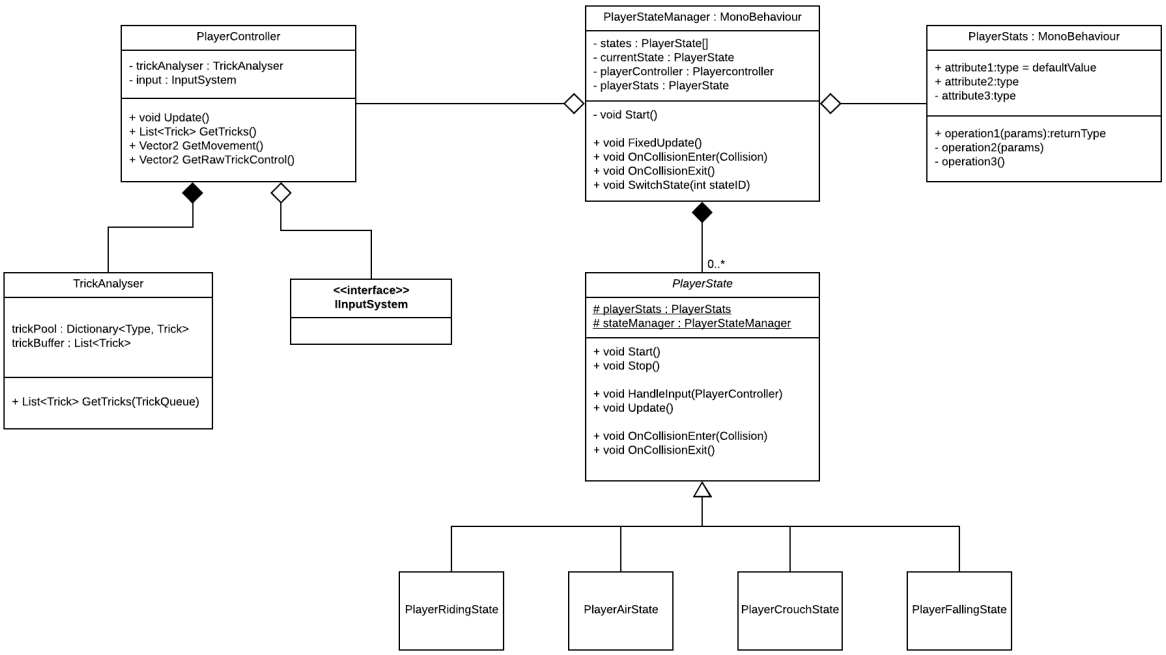
`Input, UI Input`	https://github.com/Felixbuenen/MultiPlatformDev/tree/master/Assets/Scripts/Input
`PlayerController`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Player/PlayerController.cs

Uitwerking (overig)

Tot slot licht ik kort nog wat toe m.b.t. het **player control systeem** en **trick finding**, omdat dit misschien wel de grootste uitdagingen in het project waren.

Player control

Ik heb ervoor gekozen om player control te implementeren met een State pattern. Conceptueel spreekt deze pattern me heel erg aan voor player code. Een player kan namelijk in verschillende states zitten, waarin misschien andere behaviours gelden. Zeker voor Skate L.A. heb je bijvoorbeeld te maken met een *Riding* state waarin input gelezen moet worden, maar zo heb je ook een *Falling* state waarin je dit juist niet wilt. Dit heeft uiteindelijk geresulteerd in het volgende model:



Hoewel het werkt en er enigszins logica achter dit model zit, ben ik met de uiteindelijke implementatie niet erg tevreden. Hier ga ik verder op in onder ‘Reflectie’.

Code

`Player scripts`	https://github.com/Felixbuenen/MultiPlatformDev/tree/master/Assets/Scripts/Player

Trick analysis

De trick analysis is het algoritme dat kijkt of er een trick gemaakt is en zo ja, welke. Het algoritme houdt een queue bij van de ‘trick input’ van de afgelopen 90 updates. In het geval van een PS4 controller is de trick input dus de rechter joystick. Deze buffer wordt vervolgens iedere update gecheckt, waarbij vergeleken wordt met alle bestaande tricks. De player controller vraagt vervolgens iedere update om een lijst met tricks. Als een specifieke trick die vooraan in de lijst staat uitgevoerd kan worden, dan wordt hij uitgevoerd. Zo niet, dan wordt gekeken naar de volgende trick in de lijst, net zolang totdat er een uitvoerbare trick gevonden wordt of de lijst leeg is.

Code

`Tricks`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Misc/Tricks.cs
`Trick queue (custom queue)`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Datastructures/TrickQueue.cs
`TrickAnalyzer`	https://github.com/Felixbuenen/MultiPlatformDev/blob/master/Assets/Scripts/Input/TrickAnalyzer.cs

Note: het is niet heel logisch (dit zou ik terugkijkend aangepast hebben), maar de daadwerkelijke code voor trick analyse staat in de TrickQueue klasse. TrickAnalyzer maakt hier alleen gebruik van.

Reflectie

Over het algemeen ben ik blij met het resultaat. Ik merkte dat ik gedurende het project wel steeds minder grip kreeg op m'n eigen code architectuur. Ik denk dat dit een combinatie was van niet consistent aan het project werken (dan weer een hele dag, dan weer een lange tijd niet) en onder tijdsdruk snel resultaten willen krijgen (en dus geen tijd meer nemen voor architectuur).

Code

Ten eerste ben ik erachter gekomen dat een State machine eigenlijk niet heel lekker werkt voor een player controller. Ik heb gedurende dit project veel nieuwe dingen geleerd en ben erachter gekomen dat de component-based architectuur van Unity ook meteen zijn kracht is. In mijn code heb ik dit eigenlijk ondermijnd: de hele player control architectuur in mijn game is gebaseerd op één 'manager' object, die non-MonoBehaviours aanstuurt. Het is lastig om te pinpointen waar dit nou precies mis ging, maar het komt erop neer dat het snel onoverzichtelijk werd. Wellicht kwam de verwarring ook voort uit het eerdergenoemde feit dat er te lange pauzes tussen werksessies zaten.

Daarnaast zitten er hier en daar wat 'hacks'. Deze komen voort uit snel resultaten willen en niet helemaal weten hoe bepaalde features op een elegante manier in de huidige architectuur verwerkt kunnen worden. Denk bijvoorbeeld aan het score systeem, waar ik toch sneaky een Singleton van heb gemaakt, terwijl deze eigenlijk makkelijk door de ServiceManager gemaakt had kunnen worden.

Wishlist

Er zijn ook wat features die ik had willen toevoegen, maar waar ik helaas niet aan toe ben gekomen. Zo vind ik het nog steeds interessant om een keer een platform independent achievement-systeem te bouwen. De game kan nog wat interessanter worden door wat meer tricks toe te voegen, en ook wat meer gameplay. Tot slot zou de game er een stuk beter kunnen uitzien. Hoewel ik geen artist ben (verre van) denk ik dat een beetje meer moeite op dit gebied de game een hoop kan helpen.

Mocht ik deze game verder uitwerken met de bovenstaande features, dan begin ik in ieder geval met het opschonen van de code-base!