Layout

- Grundlagen
  - RectTransform
  - Skalierung versus Resizing
  - Der Pivot Point
- Basic Layout
  - Die Anker
- Auto Layout
  - Layout Size
  - Die Komponente Layout Element
  - Layout Controller
    - Fitter Komponenten
      - Aspect Ratio Fitter
      - Content Size Fitter
    - Gruppen Komponenten
      - Horizontal/Vertikal
      - Layout Beispiele
      - Grid-Layoutgruppe
  - Flexibles Layout
    - Flexibles Spacing
      - Spacing über die Gruppe
      - Spacing über Force Expand
      - Spacer Objekte mit Layout Element
    - Flexibles Padding
      - Flexibles Padding über die Anker
      - Size mit Layoutelement überschreiben
      - Size der Childs manipulieren
      - Spacer Objekte um die Layoutgruppe herum

Grundlagen

RectTransform

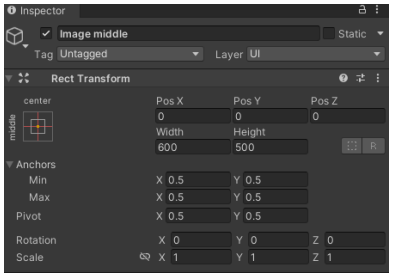


Abbildung 1: RectTransform-Komponente

Jedes GameObject unter einem Canvas erhält ein RectTransform an Stelle der normalen Transform Komponente (Abbildung). Ganz oben ist dort, wie gewohnt, die Position des Objekts. Darunter befinden sich nun Einstellungen für Breite und Höhe des Elements - denn jedes Element unter dem Canvas wird als Rechteck dargestellt. Ebenfalls wiederzufinden sind Rotation und Skalierung.

Auf die Einträge "Anchors" und "Pivot" wird später eingegangen.

Skalierung versus Resizing

Im normalen Kontext ist man gewöhnt Skalierung zu benutzen, um Objekte größer oder kleiner zu machen. Innerhalb der UI macht es mehr Sinn stattdessen Breite und Höhe anzupassen (Resizing). Das hat vor allem den Grund, das die meisten UI-Komponenten nur mit diesen Werten interagieren. Ein weiterer Anwendungsfall sind Sliced Images, wo das Resizing keinen Einfluss auf den Rand des Bildes hat - Skalierung aber schon.

Es bietet sich als Konvention an für statische Größen immer Breite und Höhe zu verwenden, während dynamische Größe über Skalierung geregelt wird (Beispiel: Animationen).

Der Pivot Point

Wenn man ein UI Objekt anwählt sieht man einen Kreis, meist in der Mitte des Rechtecks. Dieser Kreis ist der sog. "Pivot Point", das Zentrum des Elements. Seine Position ist im RectTransform mit relativen Werten zwischen 0 und 1 angegeben. Die Position (0,0) ist dabei links unten im Rechteck. Alle Transformationen - Verschieben, rotieren, skalieren, resizing - passieren ausgehend vom Pivot bzw. um ihn herum.

Dazu ein paar Beispiele:

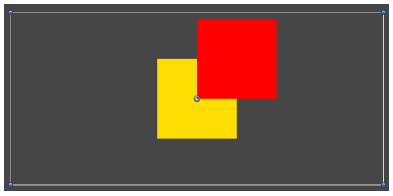


Abbildung 2: Beide Rechtecke auf gleicher Position

Das gelbe Rechteck hat den Standard Pivot von (0.5, 0.5) und das rote den Pivot links unten (0,0). Beide haben ihren Anker in der Mitte des Canvas. Bei Position (0,0,0) befindet sich ihr Pivot genau auf dem Anker. Entsprechend ist gelb genau in der Mitte, während von rot die linke untere Ecke in der Mitte ist.

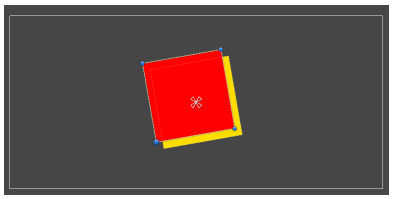


Abbildung 2: Beide Rechtecke mit gleicher Rotation in der Bildschirmmitte

Für das nächste Beispiel habe ich beide Rechtecke in die Bildschirmmitte bewegt und dann um 10 Grad auf der Z-Achse rotiert. Das gelbe Rechteck rotiert auf der Stelle. Das rote Rechteck hingegen hat sich um seine linke untere Ecke rotiert, weswegen eine Verschiebung zwischen den beiden passiert ist.

Abbildung 3: Beide Rechtecke skaliert

Ich bewege die beiden Rechtecke wieder in die Mitte und verdopple die Skalierung auf der X-Achse, um das obige Bild zu erhalten. Wenn man den Alpha-Wert des roten Rechtecks auf die Hälfte reduziert, kann man erkennen, dass es sich weiter nach rechts ausgedehnt hat, als das gelbe. Skalierung geschieht vom Pivot weg, deswegen ist das rote Rechteck mit Pivot links in Richtung rechts gewachsen. Der orangene Teil ist wo sich die beiden Rechtecke immer noch überlappen.

Übrigens: Eine Änderung der Breite auf den doppelten Wert verhält sich analog.

Manual: Basic Layout

Basic Layout

Für die einfachste Form von Layout platziert man UI Elemente relativ zum Canvas oder zu einander. Dabei werden ausschließlich die Anker benutzt, welche ein weit verbreitetes Layouting-Konzept sind.

Die Anker

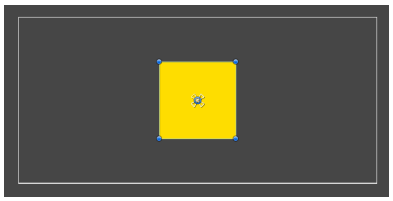


Abbildung 4: Anker und Pivot im Zentrum des Rechtecks

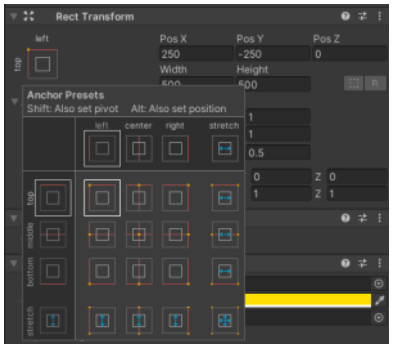


Abbildung 5: Anker-Voreinstellungen

Anker werden durch vier dreieckige Griffe im Scene View repräsentiert (Abbildung 4). Sie bestimmen zu welchem Punkt innerhalb des Parent-Rechtecks die eigene Position relativ ist – ähnlich wie beim normalen Transform, wo die Position relativ zum Ursprung des Parents ist. Ein Child ist standardmäßig zum Zentrum geankert (Abbildung 4). Häufige andere Konfigurationen können als Voreinstellungen gefunden werden, indem man auf das Anchor Preset Symbol links oben im RectTransform klickt (Abbildung 5).

Abbildung 6: Anker auf links oben

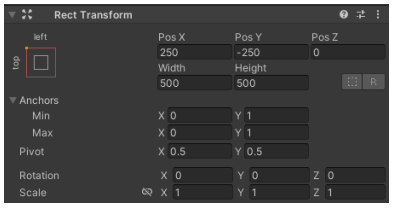


Abbildung 7: RectTransform des gelben Rechtecks

Das gelbe Rechteck aus Abbildung 6 liegt mit seiner linken oberen Ecke genau auf der linken oberen Ecke des Parents auf - diese Position ergibt sich aus der Anker-Einstellung links oben und der Verschiebung um die halbe Breite/Höhe, da der Pivot immer noch in der Mitte des Rechteckes ist.

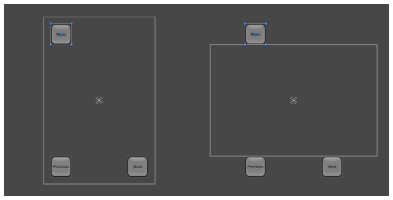
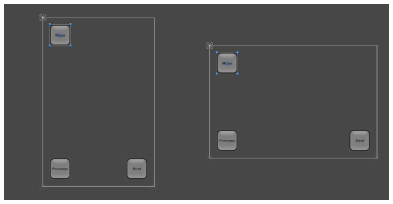


Abbildung 8a: Standard Verankerung bei Wechsel der Auflösung

Was passiert, wenn man alle Objekte einfach auf den Standardeinstellungen lässt und sie ohne sinnvolle Verankerung einfach irgendwo hinzieht, sieht man in Abbildung 8a: Auf der linken Seite ist eine gängige Portrait-Auflösung, auf der rechten Landscape - alle Buttons haben ihre Anker in der Mitte des Parent-Canvasses, weswegen sie in Landscape aus dem Canvas und damit dem Bildschirm herausragen.



Manual: Designing for multiple resolutions

Abbildung 8b: Verankerung Top-Left bei Wechsel der Auflösung

In Landscape ist das Canvas deutlich weniger hoch als in Portait, weswegen der feste Abstand zur Mitte die Buttons aus dem Canvas treten lässt. Eine sinnvollere Verankerung wäre links oben, wodurch sich der markierte Button relativ zur linken oberen Ecke positioniert

Im Inspector gibt es die Sektion „Anchors“, unter der Min und Max als 2D-Ankerpositionen gelistet sind (Mitte; Abbildung 7). "Min" beschreibt dabei die linke untere Ecke bzw. den linken unteren Anker und "Max" den rechten oberen. Die Ankerpositionen werden als Werte zwischen 0 und 1 angegeben. Solange die beiden Anker am selben Punkt liegen, besitzt das Rechteck eine fixe Breite und Höhe, während es sich relativ zum Parent positioniert.

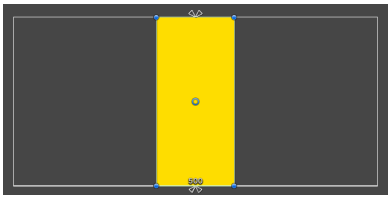


Abbildung 9: Gelbes Rechteck mit Voreinstellung stretch auf der Vertikalen

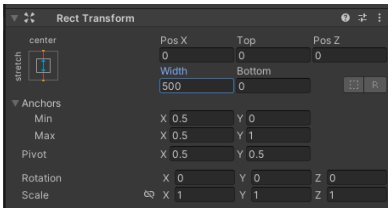


Abbildung 10: RectTransform zeigt Padding an Stelle von Position auf vertikaler Achse

Zieht man die Anker auseinander, entsteht auf der entsprechenden Achse eine relative und damit flexible Größe. In diesem Fall kann man die Werte von Min und Max als Prozente interpretieren, also 0 entspricht 0% und 1 entspricht 100%. Wenn die Anker beispielsweise jeweils auf der Ober- bzw. Unterkante des Parent-Rechtecks liegen, nimmt es immer die volle Höhe ein (Abbildung 9). Wenn der Parent schrumpft, schrumpft auch das gelbe Rechteck und das gleiche gilt auch fürs Wachsen.

Diese Konfiguration entspricht dem Preset "Strech - center" (Abbildung 5, unten). An Stelle einer absoluten Position auf der vertikalen Achse kann man nun ein Padding für "Top" und "Bottom" setzen (Abbildung 10). Man kann ebenfalls sehen, dass Anchor.Min.Y = 0% und Anchor.Max.Y = 100% sind - woraus das Stretching in der Höhe resultiert.

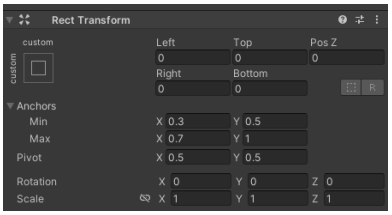


Abbildung 11: Custom Anchoring Beispiel

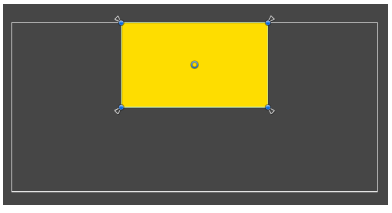


Abbildung 12: Das Rechteck hat die halbe Höhe und je 30% Padding rechts/links

Die Werte für Ankerpositionen sind übrigens frei wählbar. Man kann damit relative Layouts erstellen- wählt man z.B. die Werte wie in Abbildung 11 erhält man das Rechteck aus Abbildung 12.

Aufgeschlüsselt entsteht es so:

- Anchor.Min.X = 0.3
  Das Rechteck beginnt bei 30% des Parents. Es hat also immer 30% Freiraum zur linken Kante.
- Anchor.Max.X = 0.7
  Das Rechteck endet bei 70% des Parents. Die Gegenrichtung zum vorher gennanten. Es wird immer 30% Freiraum nach rechts gelassen.
- Anchor.Min.Y = 0.5
  Das Rechteck beginnt ab der Hälft. Also gibt es 50% Abstand zur Unterkante.
- Anchor.Max.Y = 1.0
  Das Rechteck endet auf der Oberkannte.

Ein weiteres Anwendungsbeispiel findet sich im Kapitel "Die Safe Zone".

Manual: Auto Layout

Auto Layout

Wenn Basic Layout nicht reicht kann man auf Auto Layout und die dazugehörigen Komponenten zurückgreifen. Das System Auto Layout besteht aus zwei Teilen: Layoutelemente, die im Grunde genommen ihre gewünschten Größen kennen, diese aber nicht selbst setzen und Layoutcontrollern, die schlussendlich für die Zuweisung von Breite bzw. Höhe zuständig sind.

Layout Size

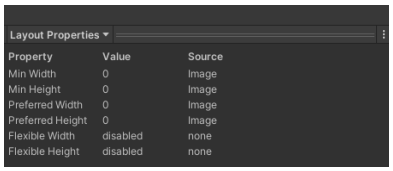


Abbildung 13: Layout Properties eines Image

Pro Dimension (Width, Height) verfügt jedes Layoutelement über drei Größen (Sizes):

1. Minimum -,
2. Preferred -,
3. Flexible Size.

Sehen kann man diese ganz unten im Inspector, unter "Layout Properties" (Abbildung 13). Manchmal ist diese Ansicht in einem Dropdown versteckt - wenn beispielsweise ein Image auf dem GameObject liegt, wird ein standardmäßig eine Vorschau vom Sprite angezeigt.

Wenn ein Layoutcontroller beginnt die Größe zuzuweisen, betrachtet er nacheinander von jedem Element diese drei Werte. Zuerst erhält jedes Element seine Minimum Size. Wenn danach noch Raum übrig ist, wird damit begonnen Elemente in Richtung ihrer Preferred Size wachsen zu lassen. Die Elemente wachsen so lang weiter, bis sie diese erreicht haben oder kein weiterer Raum mehr zur Verfügung steht. Sollte danach immer noch Raum zur Verfügung stehen, wachsen alle Elemente mit Flexible Size weiter, bis auch dieser Raum aufgebraucht ist.

Die Standardwerte für die Sizes sind 0. Während Minimum und Preferred Size tatsächliche Größenangaben sind, ist Flexible Size ein relativer Wert zwischen 0 und 1 – bedeutet also das Element nimmt gar keinen (Wert 0) oder den ganzen (Wert 1) verfügbaren Restraum ein. Sollten mehrere Elemente die gleiche Flexible Size haben, erhalten sie gleich viel Raum.

Anmerkung: Wenn mehrere Elemente eine Flexible Size haben und diese Sizes aufsummiert mehr als 1 ergeben wird die ganze Berechnung ein wenig unintuitiv. Nehmen wir zwei Layoutelemente A und B an. A hat eine Flexible Size von 1 und B eine von 4. Es steht ein Restraum von 10 Units zur Verfügung. Die zugeteilte Size ist jetzt 10 x Flexible Size / Summe aller Flexible Sizes – bedeutet also A erhält 10 x 1 / 5 = 2 units und B erhält 8 units. Ich würde daher vorschlagen darauf zu achten, dass Flexible Sizes entweder gleich groß sind oder sich zu 1 aufsummieren.

Die Komponente Layout Element

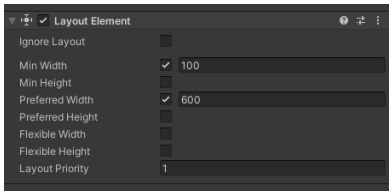


Abbildung 14: Layoutelement

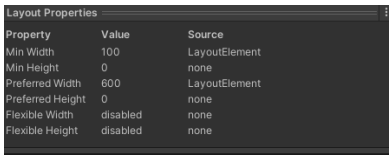


Abbildung 15: Layoutelement kontrolliert Layout Properties

Es gibt bestimmte (Layout-) Komponenten, die die Standardwerte der Sizes überschreiben (Abbildung 14). Einfache Beispiele dafür sind "Layout Element", "Image" und "Text" (Beispiele Abbildung 13 und 15). Image und Text werden die Sizes immer so setzen, dass das Element den entsprechenden Inhalt fassen kann. Wenn ein Sprite also 50 units Breite hat, setzt die Image-Komponente seine Preferred Size entsprechend.

Wann immer man Elemente in seinem Layout hat, deren Werte sich nicht automatisch ergeben oder denen man eine andere Größe als die automatische zuweisen will, benutzt man diese Komponente. Ein Anwendungsfall von mir sind "Spacer"-Objekte, welche ich für flexiblen Leerraum bzw. flexible Abstände nutze. Solche Objekte sind nichts anderes als empty GameObjects mit einem Layout Element drauf, damit sie im Layout Platz einnehmen.

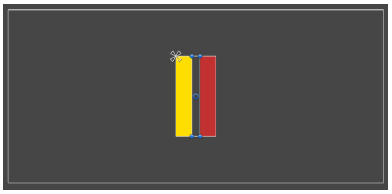


Abbildung 16: Leeres GameObject als flexibler Leerraum

Auf Abbildung 16 ist ein leeres Objekt im Layout zwischen den beiden Rechtecken. Das LayoutElement hat Min Width = 20 und Preffered = 50. Da momentan nicht genug Platz ist, bleibt es bei 20 units - sollte die Gruppe aber größer werden, erhöht sich der Abstand entsprechend.

Neben den Sizes bietet das Layout Element noch zwei weitere Optionen:

- **Ignore Layout,**
- **Layout Priority.**

Ignore Layout ist ein Toggle und kommuniziert an Layoutcontroller auf dem Parent, dass dieses Objekt nicht teil des Layouts ist. Das ist nützlich, um Objekte innerhalb einer Layoutgruppe oder in Relation zu dieser zu platzieren, ohne das Layout zu beeinflussen.

Layout Priority legt die Priorität dieser Layout-Komponente gegenüber anderen fest. Das bedeutet, dass immer die Sizes von dem Element mit der höchsten Priorität an Layoutcontroller kommuniziert werden. Eine Layoutgruppe hat beispielsweise die Priorität 0 (steht als Property im Source Code). Wenn man jetzt auf dem gleichen GameObject ein Layout Element hinzufügt, dass standardmäßig mit Priorität 1 startet, überschreibt dieses die Sizes der Gruppe. Setzt man die Priorität des Layout Elements auf -1, ist die Gruppe wieder priorisiert. Sind mehrere Komponenten mit der gleichen Priorität aktiv, wird immer der höchste Wert von allen gewählt.

Layout Controller

Layout Controller kontrollieren die Size und/oder Position von einem oder mehreren Layoutelement(en). Es gibt zwei Sorten von Layout Controllern:

- Fitter,
- Gruppen.

Fitter Komponenten

Fitter kontrollieren immer ihr eigenes Layout Element, also das Objekt, auf dem sie platziert sind.

Aspect Ratio Fitter

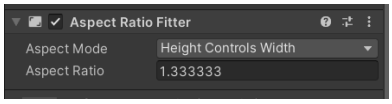


Abbildung 17: Aspect Ratio Fitter Komponente

Der Aspect Ratio Fitter verändert das Element so, dass es ein bestimmtes Seitenverhältnis einhält. Dafür orientiert er sich, je nach Modus, an einem eigenen Maß oder beiden Maßen seines Parents. Für diese Orientierung gibt es verschiedene Modi:

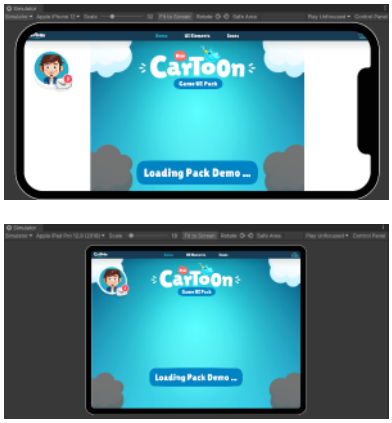


Abbildung 18: Aspect Ratio Fitter aus Abbildung 15 kontrolliert ein 4:3 Hintergrund auf iPhone 12 und iPad 12.9

Height controls width
Die Höhe des Objektes bestimmt entsprechend des Seitenverhältnisses die Breite. Das Hintergrund in Abbildung 18 ist auf "stretch" gestellt – somit nimmt es auf dem Pad den gesamten Bildschirm ein, da es das auf dem Fitter eingetragene Seitenverhältnis (siehe Abbildung 17) exakt trifft. Auf dem Phone hingegen reicht die resultierende Breite nicht aus um den Bildschirm zu füllen.

Abbildung 19: Aspect Ratio Fitter im Modus "Width" auf iPhone 12

Width controls height

Die Breite des Objektes bestimmt entsprechend des Seitenverhältnisses die Höhe. Die resultierende Höhe ist auf dem Phone wesentlich zu groß, da es ein 16:9 Seitenverhältnis hat – die Wolken verschwinden vom Bildschirm. Auf dem Pad bleibt die Ansicht aus Abbildung 18 bestehen.

Fit in Parent

In diesem Modus wächst das Element gleichmäßig auf beiden Achsen unter Einhaltung des Seitenverhältnisses. Sobald eine Achse mit der des Parents gleich zieht wird gestoppt. Je nach dem, wie gut das gewählte Seitenverhältnis zu dem des Parents passt, kann es sein, dass das Element diesen nicht komplett ausfüllt. Im Beispiel erzeugt diese Einstellung identische Ergebnisse zu "Height controls width" bzw. Abbildung 18.

Envelope Parent

Ähnliches Verhalten wie "Fit in Parent". Allerdings wird hier gewachsen, bis beide Achsen gleichziehen. Das führt dazu, dass das Element den Parent komplett ausfüllt und gegebenenfalls auf einer Achse über steht. Für das Beispiel ergibt sich das gleiche Ergebnis wie mit "Width controls heigth" bzw. in Abbildung 19. Auf dem Phone wird sehr früh die maximale Höhe erreicht und anschließen überschritten, während auf die maximale Breite angewachsen wird, weswegen das gelbe Rechteck auf der Vertikalen weit über steht.

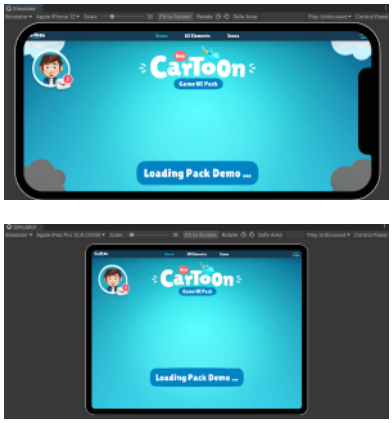


Abbildung 20: Beispiel für Envelope Parent mit 16:9 Ladebildschirm auf Phone und Pad

Ich habe den Aspect Ratio Fitter z.B. für einen Ladebildschirm genutzt. Dazu nutze ich den Modus Envelope Parent und trage das Seitenverhältnis der Grafik in den Fitter ein. Es wird immer der ganze Bildschirm ausgefüllt, allerdings sieht man unterschiedliche Ausschnitte vom der Grafik, je nach dem, wie gut das Seitenverhältnis zum Gerät passt. Auf dem Pad sehen wir beispielsweise an den Seiten weniger, weswegen die Wolken verborgen bleiben.

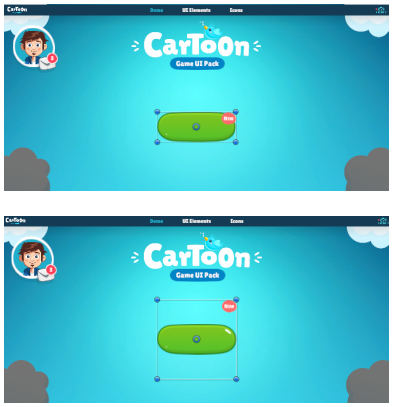


Abbildung 21: Control Modi vs preserve aspect

Mit den Control Modi für Width/Height kann man ein ähnliches Verhalten wie die Funktion "Preserve Aspect" auf der Image-Komponente erhalten. Der wichtige Unterschied dabei ist, dass das Bild auch tatsächlich so groß wird, wie es das Seitenverhältnis vorgibt und nicht nur so aussieht. Das erlaubt einem dann korrekte Verankerung am Image. In Abbildung 21 kann man entsprechend sehen, dass das "New" Icon im zweiten Bild falsch verankert ist, obwohl die Grafiken gleich groß aussehen.

Größtes Problem des Aspect Ratio Fitters ist, dass man ihn nur nutzen kann, wenn das Element nicht von einer Layoutgruppe gesteuert wird. Ein Problem was sich auch bei der nächsten Komponente fortsetzt.

Content Size Fitter

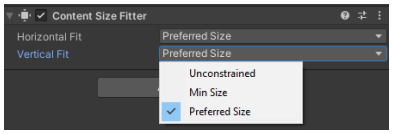


Abbildung 22: Die Komponente Content Size Fitter

Der Content Size Fitter setzt die Maße des Elements basierend auf einem Layout Component auf dem gleichen Objekt. Das können beispielsweise Layoutgruppen, Images oder Text sein. Er verfügt über folgende Optionen, die jeweils auf die horizontale oder vertikale Achse angewendet werden können:

- **Unconstrained**
  Der Content Size Fitter hat keinen Effekt auf diese Achse.
- **Min Size**
  Das Element wird vom Content Size Fitter immer auf seine minimale Größe gesetzt.
- **Preferred Size**
  Das Element wird immer genau auf die Preffered Size gesetzt. Wenn keine Preferred Size angegeben ist, dafür aber eine Min Size, wird dieses stattdessen verwendet.

Abbildung 23: Das Rechteck fast den Text genau ein

Für mich leistet der Content Size Fitter seine Arbeit vor allem auf Root Elementen von Panels, um automatisch die richtige Größe einzustellen. Eine weitere gute Anwendung ist in Kombination mit Text, wo der Content Size Fitter den Text Container wachsen lässt um den Text bei gleicher Schriftgröße einzufassen (Abbildung 23). Er ist ebenfalls sehr nützlich für die Gridlayoutgruppe.

Gruppen Komponenten

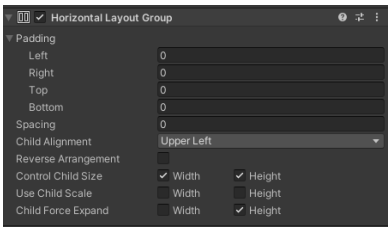


Abbildung 24: Horizontale Layoutgruppe

Gruppen kontrollieren Positionen ihrer Childs. Wenn gewünscht können sie ebenfalls die Größen kontrollieren - wie das genau abläuft wird im Unterkapitel "Layout Beispiele" erklärt. So oder so gilt: Sollte die Gruppe nicht groß genug sein, um ihre Childs zu fassen, beginnen diese aus ihr herauszuragen.

Jede Layoutgruppe hat folgende Optionen:

- **Padding**
  Innerer Rand der Gruppe aus vier Richtungen,
- **Spacing**
  Abstand zwischen den Childs,

Man kann auch negative Werte nutzen, was die Größe der Gruppe verringert. Damit sich die Childs überlappen, fügt man negatives Spacing hinzu. Bei negativen Padding ragen sie dann aus der Gruppe heraus.

- **Child Alignment**
  Bestimmt Verankerung der Childs innerhalb der Gruppe.

Diese Einstellung beeinflusst vor allem, in bzw. aus welcher Richtung sich die Gruppe füllt. Mit Alignment "Center ..." bleiben die kontrollierten Elemente beispielsweise zentriert.

Eine Layoutgruppe fungiert selbst als Layoutelement. Ihre Größe entspricht der Summe der Größe ihrer Childs. Sind in einer Gruppe beispielsweise drei Elemente mit Minimum Width 50, so hat die Gruppe eine Minimum Width von 150.
Padding bzw. Spacing werden ebenfalls noch dazugerechnet. Hat die Gruppe zusätzlich noch ein Spacing von 30, ist ihre Minimum Width 210 (150 von den Childs plus zwei Spacings von 30 zwischen den drei Childs).

Die berechnete Größe wird dann auch an Layout Controller auf dem Parent der Gruppe oder an Fitter auf dem gleichen Objekt weitergegeben. Entsprechend kann man Layoutgruppen auch schachteln.

Es ist wichtig, dass eine Gruppe groß genug ist, um all ihre Childs einzufassen. Sollte eine Achse zu klein sein, beginnen Childs aus der Gruppe herauszuragen. Am einfachsten kann man das über Fitter Komponenten wie den Content Size Fitter verhindern, welche die Layout-Informationen der Gruppe nutzen, um ihre Maße entsprechend ihrer Childs anzupassen.

Horizontal/Vertikal

Horizontale bzw. vertikale Gruppen platzieren ihre Childs nebeneinander auf der entsprechenden Achse.

Bei diesen Gruppen gibt es noch weitere Optionen:

- **Reverse Arrangement**
  Kehrt die Reihenfolge der Childs um.

Ist vor allem nützlich, wenn die Sortierung von der Reihenfolge abweicht - da in der UI immer das unterste Objekt vorne angezeigt wird und sich die Gruppe immer von oben nach unten füllt. Möchte man das vorn angezeigte Objekt im Layout an erster Stelle haben, nutzt man "Reverse Arrangement".

Basierend auf der eigenen Größe probiert die Gruppe ihren Childs die entsprechende Menge an Minimum-/Preffered-/Flexible Size zuzuteilen. Wie genau diese Verteilung ausfällt wird von den drei Kontroll-Optionen beeinflusst:

- **Control Child Size**
  Kontrolliert die Gruppe die Größe ihrer Childs?

Wenn ja - wird den Childs auf der entsprechenden Achse Minimum-/Preffered-/Flexible Size zugeteilt. Nehmen wir einmal folgendes Beispiel an: Unsere Layoutgruppe hat eine feste Größe von 300px, ohne Spacing und Padding. Unter ihr befinden sich drei Elemente mit Min Size 50 und Preferred Size 200. Zunächst wachsen alle Elemente auf ihre Minimum Size von 50, womit 150px der Gruppe belegt sind. Da alle Elementen eigentlich 200px groß sein wollen, kann jedes von ihnen noch wachsen. Die restlichen 150px werden also der entsprechenden Priorität nach auf die Elemente verteilt. Somit sind schlussendlich alle Elemente 100px groß.
Wenn nein - wird nur die Position der Childs durch die Layoutgruppe gesetzt.

- **Use Child Scale**
  Soll die Gruppe bei Zuweisung von Size und Position die Skalierung der Elemente beachten?

Beispiel: Ein Element hat Min Size 100, aber einen Scale von 0.5 auf der entsprechenden Achse.

Wenn ja - Die Layoutgruppe behandelt das Element mit seiner echten Größe von 50.
Wenn nein - Das Element wird so behandelt, als hätte es trotz Scaling seine volle Größe von 100.

- **Child Force Expand**
  Sollen die Childs die entsprechende Achse voll ausfüllen?

Bei dieser Option dreht es sich darum, was mit freiem Platz geschehen soll, der von keinem Element beansprucht wird.

Wenn nein - Der freie Platz bleibt leer.
Wenn ja - Die Elemente werden so verändert, dass sie den Raum voll ausfüllen. Wenn auf der entsprechenden Achse auch die Size kontrolliert wird wachsen alle Elemente, bis die Gruppe ausgefüllt ist.
Falls die Size nicht kontrolliert wird, entsteht Leerraum zwischen den Elementen.

Layout Beispiele

Jetzt mal ein paar Beispiele.

Hierarchie:

- **Gruppe**
  Size: 100
  - **Element A**
    Minimum Size: 10
    Preffered Size: 50
    Flexible Size: 0
  - **Element B**
    Minimum Size: 30
    Preffered Size: 0
    Flexible Size: 0

Zunächst erhalten Element A und B ihre Minimum Size von 10 bzw. 30. Danach ist noch Raum von 60 units übrig. Element B verfügt über keine weiteren Size Einstellungen und ist entsprechend fertig. Element A hat eine Preferred Size von 50 und möchte damit noch um weitere 40 Units wachsen. Da dieser Raum noch verfügbar ist, kann Element A seine Preferred Size erreichen und ist nun 50 units groß. Es bleiben 20 units als Leerraum im Layout zurück, da diese nicht beansprucht wurden.

Was ist aber nun, wenn beide Elemente eine Flexible Size haben?

Hierarchie:

- **Gruppe**
  Size: 100

        - **Element A**
          Minimum Size: 10
          Preffered Size: 50
          Flexible Size: 1
        - **Element B**
          Minimum Size: 30
          Preffered Size: 0
          Flexible Size: 1

Das Verhalten bleibt bis zum Zuweisen der Flexible Size zunächst einmal gleich. Danach werden die Flexible Size Werte betrachtet. Diese sind gleich groß, also erhalten beide Elemente die Hälfte der verbleidenden 20 units. Element A erreicht so eine Größe von 60 units. Element B wächst auf 40 Units heran.

Nehmen wir als nächstes an, dass wir die Flexible Size ungleich verteilen wollen:

Hierarchie:

- **Gruppe**
  Size: 100
    - **Element A**
      Minimum Size: 10
      Preffered Size: 50
      Flexible Size: 0.25
    - **Element B**
      Minimum Size: 30
      Preffered Size: 0
      Flexible Size: 0.75

In diesem Fall erhält A genau ein Viertel, also 5, zusätzliche units und B die restlichen drei Viertel, entsprechend 15 units.

Im letzten Beispiel reicht die Size der Gruppe nicht aus, um beide Elemente auf ihre gesamte Preferred Size anwachsen zu lassen:

Hierarchie:

- **Gruppe**
  Size: 100
    - **Element A**
      Minimum Size: 10
      Preffered Size: 50
      Flexible Size: 0
    - **Element B**
      Minimum Size: 30
      Preffered Size: 60
      Flexible Size: 0

In diesem Fall wachsen beide Elemente gleichmäßig, bis der Raum vollständig aufgebraucht ist. Element A endet hier bei in etwa 44 units. Element B bei 56. Beide Elemente sind ähnlich weit von ihrer Preferred Size entfernt – Element B

ist näher dran, weil die Differenz zwischen Minimum Size und Preferred Size bei ihm geringer ausfiel.

Grid-Layoutgruppe

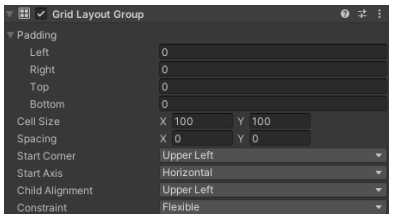


Abbildung 25: Grid-Layoutgruppe

Eine Gridlayoutgruppe platziert Childs auf einem Grid. Sie verfügt über folgende zusätzliche Optionen:

- **Cell Size**
  Größe der einzelnen Grid-Zellen.

Diese Layoutgruppe ignoriert Size-Einstellungen der enthaltenen Elemente. Stattdessen wird jedem Child die fixe Zellengröße zugeteilt.

- **Start Corner**
  Ecke in der sich das erste Element befindet.
- **Start Axis**
  Die primäre Achse für das platzieren von Elementen.

Die Starteinstellungen sind maßgeblich dafür, wie sich das Grid füllt. Start Corner bestimmt den Ursprung. Start Axis kann horizontal oder vertikal sein und bestimmt die Tendenz zuerst Reihen bzw. Spalten zu füllen, bevor eine neue Spalte bzw. Reihe angefangen wird. Wann eine Zeile bzw. Spalte voll ist, wird durch die Breite bzw. Höhe des Gruppen Elements bestimmt.

- **Constraint**
  Limitiert das Grid auf eine feste Anzahl an Zeilen bzw. Spalten.

Es gibt drei Einstellungen für Constraint: Flexible, Fixed Row- und Fixed Column Count. Flexibel kann man als eine Art ausgeglichene Einstellung verstehen. Reihen und Spalten werden so zugeordnet, dass die Childs in die Gruppe passen. Dabei wird versucht eine ungefähr gleiche Anzahl an Reihen und Spalten zu erhalten.

Allgemein kann es viele mögliche Kombinationen von der Anzahl an Reihen und Spalten geben, welche die Zellen in die Layoutgruppe einfassen. Mit Hilfe von Start Axis und den Row- bzw. Column Count Constrains wird beeinflusst, welche Kombination ausgewählt wird. Das ist vor allem wichtig, wenn man in Kombination mit anderen Auto Layout-Komponenten wie z.B. dem Content Size Fitter arbeitet.

Zur Erinnerung: Auto Layout berechnet Höhe und Breite unabhängig von einander, aber innerhalb eines Grids hängt die Anzahl an Zeilen mit der Anzahl an Spalten zusammen und umgekehrt. Um also korrekt flexible Grids zu erstellen bieten sich folgende Kombinationen an:

- **Grid mit flexibler Breite und fester Höhe**
  Fixed Row Count aktiv, Content Size Fitter mit Horizontal Fit – Preffered Size

In dieser Konfiguration wird das Grid horizontal wachsen, wenn mehr Elemente hinzugefügt werden. Vertical fit auf dem Content Size Fitter sollte ebenfalls auf Preffered Size gesetzt sein, da man sich ansonsten selbst um die Zuweisung einer korrekten Höhe kümmern muss.

- **Grid mit flexibler Höhe und fester Breite**
  Fixed Column Count aktiv, Content Size Fitter mit Vertical Fit – Preffered Size

Dieses Grid wird vertikal wachsen, wenn mehr Elemente hinzugefügt werden. Erneut: Horizontal fit auf dem Content Size Fitter sollte ebenfalls auf Preffered Size gesetzt sein, da man sich ansonsten selbst um die Zuweisung einer korrekten Breite kümmern müsste.

- **Voll flexibles Grid**
  Flexible aktiv, Content Size Fitter mit Horizontal Fit und Vertical Fit auf Preffered Size

Man kann ebenfalls auf beiden Achsen flexibel wachsen, bestimmt durch Start Axis. Allerdings verliert man die Kontrolle über die Anzahl an Zeilen und Spalten, welche ungefähr gleich verteilt werden.

Der vielleicht größte Nachteil der Gridlayoutgruppe für die Mobile Plattform ist die fixe Größe der Zellen. Es kann keine unterschiedlich breiten oder hohen Zellen geben. Natürlich lässt sich das Ganze trotzdem auf unterschiedliche Wege erreichen:

- **Das Grid als eine Art Gestaltungsraster**

Anstatt die Gridzellen direkt mit Inhalt zu füllen können sie auch als Parent für Elemente dienen, die dann frei in der Wahl ihrer Größe sind bzw. ihre Größe immer noch abhängig von der Zellengröße machen können. Beispielsweise kann innerhalb einer Gridzelle auch eine Layoutgruppe geschachtelt sein – entweder Horizontal/Vertikal oder ein weiteres Grid.

- **Schachteln von Layoutgruppen zur Erzeugung des Grids**

Im weitesten Sinn ist die Gridlayoutgruppe nur eine praktische Hilfe für Grids mit einer dynamischen Anzahl an Elementen. Ist die Anzahl an Elementen jedoch statisch bzw. zur Designzeit bereits bekannt, so kann man ein Grid ebenfalls aus verschachtelten Layoutgruppen bauen. Beispielsweise eine horizontale Layoutgruppe als einzelne Zeile, unter der dann vertikale Gruppen als Spalten platziert werden.

Da die fixe Größe der Zellen wie gesagt nicht von Auto Layout, also den Sizes, beeinflusst wird, kann . Aufgrund dessen würde ich erneut das Schachteln der anderen Layoutgruppen bevorzugen, da man sich ansonsten mit eigenem Code zum Setzen der Zellengröße oder einer eigenen Gridlayoutgruppe behelfen muss.

Flexibles Layout

Dieser Abschnitt ist eine Art "How-To" mit verschiedenen Techniken für flexible Abstände.

Flexibles Spacing

Ich verwende drei verschiedene Wege um in meinen Layouts Abstände einzubauen:

- Spacing Option der Layoutgruppe
- Force Expand Option der Layoutgruppe
- Spacer Objekte mit Layout Element

Auf die Anwedungsgebiete, sowie Vor- und Nachteile möchte ich im Folgenden eingehen.

Spacing über die Gruppe

Der einfachste Weg ist es einen Wert für Spacing auf der Gruppe zu setzen.

Vorteil:

- Dieser Wert ist fix - Auto Layout wird ihn immer einhalten. Er wird ebenfalls bei der Berechnung der Size mit einbezogen und somit auch an Fitter Komponenten kommuniziert.

Nachteil:

- Dieser Wert ist fix. Wenn die Gruppe schrumpft, rücken Elemente nicht näher zusammen - es kann also vorkommen, das nun nicht mehr genug Platz für alle vorhanden ist und Elemente aus der Gruppe ragen.

Spacing als fixer Wert funktioniert am besten, wenn zuvor bereits absehbar ist, dass es problemlos eingehalten werden kann. Um dem Problem der fehlenden Flexibilität entgegenzuwirken, könnten die Childs der Gruppe flexible Größen haben, um eventuell zu schrumpfen oder zu wachsen - allerdings stellt sich hier die Frage, ob man festen Abständen so viel Priorität einräumen will.

Wichtig: Diese Spacing Variante unterstützt als einzige negative Werte, wodurch man Elemente überlappen lassen kann.

Spacing über Force Expand

Zur Wiederholung: Wenn auf einer Layoutgruppe die Option Child Force Expand angewählt ist, werden die Childs so verteilt, dass sie die Gruppe maximal ausfüllen. Wird die Größe der Childs nicht zusätzlich von der Gruppe kontrolliert, kann Leerraum entstehen, wenn die Gruppe größer als die Summe ihrer Childs ist.

Vorteil:

- Die Menge an Leerraum ist gleichverteilt und flexibel. Sie ist abhängig vom Größenverhältnis der Childs zur Gruppe.

Nachteil:

- Es gibt kein Minimum für die Abstände. Wenn die Gruppe schrumpft, schrumpfen auch die Abstände. Sollte die Gruppe ihre Childs beispielsweise exakt einfassen oder gar zu klein sein, gehen die Abstände gegen Null,
- Die Option "Control Child Size" kann nicht verwendet werden, da die Childs so den gesamten Raum einnehmen.

Auf diese Art und Weise lässt sich unkompliziert dynamisches Spacing erzeugen - allerdings nur bis zu einer gewissen Mindestgröße der Gruppe. Ausschlaggebend für die Richtung des Spacings ist die Option Child Alignment:

Anzuwenden ist diese Technik, wenn auf der Achse des Spacings keine dynamische Größe existiert - zum Beispiel eine Gruppe von Bildern mit fixer Höhe, welche dynamischen Abstand auf der horizontalen hat. Die Nachteile könnte man beispielsweise ausbessern, indem die Gruppe selbst eine ausreichende Minimum Size hat, durch die dann ein Mindestmaß an Spacing gewährleistet ist.

Spacer Objekte mit Layout Element

Man kann ebenfalls ganze, quasi leere (sprich ohne Grafik) Objekte für Spacing benutzen. Diese enthalten dann nur ein Layoutelement, was der Gruppe ihre Size mitteilt.

Vorteil:

- Die Menge an Leerraum ist flexibel - sowohl mit Maximum als auch Minimum,
- sie kann zwischen jedem Objekt der Gruppe individuell angepasst werden,
- Spacer können Prefabs sein und so Abstände über mehrere Panels synchronisieren,
- man kann Elemente an den Spacern verankern.

Nachteil:

- Neue Elemente können nicht mehr dynamisch in die Gruppe eingefügt werden, es sei denn, man fügt auch weitere Spacer hinzu,
- erhöht die Anzahl an GameObjects, was Hierarchien komplizierter gestaltet und mehr Performance kostet,
- kann Iterationsgeschwindigkeit verringern, wenn man nicht mit Prefabs arbeitet,
- erhöht Komplexität von Hierarchie und ggf. Projekt.

Diese Variante gibt einem die meiste Flexibilität und Kontrolle, bringt aber auch mehr Komplexität mit sich. Diese Komplexität kann sich auf das gesamte Projekt ausweiten, wenn man Spacer als Prefabs verwendet. In der Theorie

könnte man ähnlich wie in UX-Tools das Spacing in mehreren Elementen der UI mit nur einer Änderung auf dem Prefab anpassen. Das klingt erstmal sehr praktisch, kann aber auch gleichermaßen tückisch sein - es wird viel Achtsamkeit im Umgang mit solchen Strukturen gebraucht. Genau deswegen ist es in den meisten Fällen leichter auf die zuvor genannten einfacheren Optionen zurückzugreifen.

Eine wirklich gute Anwendung ist das synchronisieren der Abstände über mehrere Panels. Nehmen wir an, dass unser Spiel ein HUD besitzt, was stehts am oberen Bildschirmrand ist. Andere Elemente auf der gleichen Ebene sollten einen Abstand zum HUD wahren, um dieses nicht zu überlappen. Natürlich kann jedes weitere Element nun schlicht einen festen Abstand nach oben aufweisen, welcher dann die Höhe des HUDs und eventuelles Padding ist - was passiert aber, wenn das HUD nun durch ein neues Design seine Höhe ändert? Man muss loslaufen und diesen Wert auf jedem anderen Element aktualisieren. Um diese Arbeit zu sparen, kann schlicht ein Spacer Prefab erstellt werden, welches die selben Maße wie das HUD aufweist. Jetzt muss bei einer Änderung des Designs nur noch das Spacer Prefab angepasst werden und nicht jedes einzelne Objekt. Diese Technik hilft ebenfalls, wenn das HUD eine flexible Höhe hat, z.B. für den Porträt-Modus.

Flexibles Padding

Ähnlich wie beim Spacing kann manchmal auch beim Padding ein fixer Wert nicht ausreichend sein. Hierfür gibt es folgende Lösungen:

- Anker Positionen entsprechend wählen
- Size der Layoutgruppe mit Layoutelement überschreiben
- Size der Childs manipulieren
- Spacer Objekte um die Layoutgruppe herum

Flexibles Padding über die Anker

Vorteil:

- Funktioniert ohne Auto Layout

Nachteil:

- Kann nicht auf Childs einer Layout Gruppe gesetzt werden,
- nur inneres Padding möglich,
- keine Obergrenze.

Indem man die Ankerpositionen korrekt wählt, kann man ebenfalls ein Padding relativ zur Größe des Parents erreichen.

Eine beispielhafte Konfiguration wäre Min (0.25,0) und Max (0.75, 1). Auf diese Art und Weise erhält man ein Padding von 25% auf der X-Achse, jeweils links und rechts. Da das Padding immer relativ zum Parent ist, kann man leider keinen maximalen Wert für die Menge an Padding festlegen. Es bedeutet ebenfalls, dass das Padding nach Innen ist.

Size mit Layoutelement überschreiben

Zur Erinnerung: Bei Layoutcontrollern gibt es Priorisierung. Das Layoutelement hat die höchste Priorität und kann somit die Size der Layoutgruppe überschreiben. Wenn man dann die Layoutgruppe größer werden lässt, als deren Children das benötigen, entsteht Leerraum um die Gruppe herum - vorausgesetzt das die Option Child Force Expand auf der entsprechenden Achse nicht genutzt wird. Wo der Leerraum entsteht ist vom Alignment der Childs abhängig

[Beispiele]

Vorteil:

- Das Padding wird immer noch auf dem gleichen GameObject kontrolliert, auf dem sich die Layoutgruppe befindet.

Nachteil:

- Wenn sich durch neues Design die Werte der Children ändern, muss man per Hand nachrechnen und eintragen - normalerweise wäre das automatisch passiert,
- kein Force Expand mehr möglich, wenn Padding auf der Achse vorhanden sein soll.

Der eine Vorteil dieser Methode mag unscheinbar klingen, es ist jedoch nicht zu verachten, dass in dieser Variante die Padding-Einstellung noch gut nachvollziehbar ist und nicht so versteckt (wie bei der folgenden Variante).

Size der Childs manipulieren

Diese Methode funktioniert nur mit Images oder wenn Spacer Objekte innerhalb der Gruppe genutzt werden. Die Idee ist, dass wir Objekte größer werden lassen, ohne das optisch mehr Raum mit Grafik gefüllt wird. Wenn wir z.B. Spacern in einer Horizontalen Gruppe eine andere Höhe als die restlichen Elemente geben entsteht Padding auf der vertikalen Achse. Umgekehrtes funktioniert für die Vertikale Layoutgruppe - andere Breite für Spacer führt zu Padding auf der Horizontalen. Wenn wir Padding auf der kontrollierten Achse erzeugen wollen, fügen wir einfach am Anfang und Ende der Gruppe ein Spacer Objekt mit der Size des Paddings ein.

Ohne Spacer Objekte kann man das Ganze auch mit Images und der Option "Preserve Aspect Ratio" erreichen, da diese nicht mit Auto Layout kommuniziert. Allerdings ist man so auf eine Achse beschränkt, da die Images ansonsten wieder im Sinne der Aspect Ratio wachsen würden. Das ist aber eher ein Hack, als das ich es wirklich für gut empfinden würde.

Vorteil:

- Die Layoutgruppe berechnet ihre Maße weiterhin automatisch,
- Spacer Prefabs können Iterationszeit sparen.

Nachteil:

- Padding wird in gewisser Hinsicht versteckt, da es implizit entsteht,

- kein Force Expand mehr möglich, wenn Padding auf der Achse vorhanden sein soll,
- erfordert Spacer Objekte,
- Hacky.

Ich glaube eigentlich nicht, dass diese Methode empfehlenswert ist. Was man sich hier potenziell an Iterationszeit sparen kann wird durch die gestiegene Komplexität zunichte gemacht.

Spacer Objekte um die Layoutgruppe herum

Bedenke: Wir können Layoutgruppen schachteln. Eine Layoutgruppe mit flexiblem Padding wäre dann also drei Objekte unterhalb einer weiteren Layoutgruppe - enstprechend ein Spacer, Layoutgruppe und ein weiterer Spacer.

Vorteil:

- Die Layoutgruppe berechnet ihre Maße weiterhin automatisch,
- Spacer Prefabs können Iterationszeit sparen,
- Padding ist unabhängig vom Alignment der Childs,
- erlaubt Force Expand.

Nachteil:

- Komplexere Hierachie,
- Padding ist nur auf der von der Obergruppe kontrollierten Achse möglich.

Diese Variante bietet sich vor allem an, wenn die Layoutgruppe sowieso schon unterhalb einer anderen Gruppe liegt (Beispiel: Mehrere Zeilen Inhalt innerhalb Panels, also horizontale Gruppen unter einer vertikalen). Vielleicht will man auch andere Alignment Optionen nehmen oder Force Expand nutzen. Leider kann man wie gesagt nur auf der kontrollierten Achse Padding erzeugen - auf der anderen geht das nur mit einer der beiden zuvor beschriebenen Methoden.