# Skærmstørrelser

## Indledning

Når man skal udvikle en brugergrænseflade er der flere elementer, som skal tages højde for bl.a. skærmstørrelse og dimensionerne, altså forholdet mellem længden og bredden. Skærmstørrelsen og dimensionerne på kan variere betydeligt alt efter, hvilken platform, der udvikles til, herunder mobile enheder som smartphones og tablets, men også computerskærme, tv mv. I dette projekt, hvor produktet er henvendt til smartphones og tablets, er der også et væld af forskellige skærmstørrelser, som skal indregnes i udviklingen. Konsekvensen af ikke at indarbejde de nødvendige fleksible størrelser kan i værste tilfælde gøre produktet uanvendeligt, da brugeren ikke kan benytte sig af de funktioner, som forventet. I bedste tilfælde er brugergrænsefladen ikke æstetisk nydelig, men dog anvendelig.

## Analyse af skærmstørrelser

Før udviklingen påbegyndes er det nødvendigt at vide præcist, hvilke skærmstørrelser er mest populære og derved gør sig gældende. Samtidig gør dette det også muligt at spare ressourcer, hvis der ikke behøves at udvikles og testes til småt udbredte skærmstørrelser.

Som kan ses ud fra denne graf, ligger størstedelen af de mobile enheder inden for relativt få forskellige skærmstørrelser, da ’Andre’ kun dækker over 16,8% af alle enheder.

Det, som dog er mere relevant at redegøre for, er dimensionerne på skærmen. Da det samme størrelsesforhold betyder, at det er muligt kun at nøjes med at skalere brugergrænsefladen, volder dette kun få problemer. Derimod er det ikke muligt kun at skulle skalere grænsefladen, hvis dimensionerne er anderledes.

Kigger man derfor på de typiske dimensioner på markedet, ser man at ’16:9’ og ’5:3’ udgør knap 50% af markedet. Da disse udgør de ydre ekstremer af de mest brugte dimensioner, er det derfor muligt at lægge fokus her.

## Brugergrænseflade i Unity3D

I dette projekt har det været nødvendigt at gøre brugergrænsefladen dynamisk til forskellige skærmstørrelser, hvilket har været i fokus under udviklingen af menusystemet. Systemet, som består af en række GUI-elementer, er bygget i Unity3D’s eget miljø vha. scripting. Miljøet minder meget HTML og ASP, men mangler noget funktionalitet i form af relative koordinater på skærmbilleder, som er under opsejling en ny opdatering.

Menuen, som i første omgang var udviklet ved brug af faste koordinator på skærmen samt faste pixelstørrelser, var ikke optimal til forskellige skærmstørrelser. For at overkomme dette problem er der gjort stort brug af script-funktionerne Screen.Height og Screen.Width, som henviser til skærmens længde og bredde. Er skærmen derfor 800x480, kan man finde midten af skærmen ved at skrive ’Screen.Width / 2’ i scriptet.

Et eksempel på en knap kan derfor se sådan ud:

GUILayout.Button("New Game / Continue", GUILayout.Height(Screen.height / 12));

GUILayout opretter et element, som automatisk bliver lagt i rækker på skærmen, som gør koden mere overskueligt. Derefter bliver der oprettet en knap via ’Button’, som indeholder følgende parametre, tekst, højde og vidde. Teksten henviser til den tekst, som vises på selve knappen på i brugergrænsefladen, hvor i dette tilfælde er ”New Game / Continue”, som er det første punkt hovedmenuen. De næste variabler, højde og vidde, henviser til størrelsen på selve knappen, hvor højden i dette stykke kode er skærmens højde divideret med 12. 12 er her valgt udelukkende ud fra, hvad der så passende ud fra udviklerens side. Vidden er ikke angivet her, da ’GUILayout’ automatisk henter højden og vidden fra parent-objektet, medmindre andet er angivet.

Det parent-objekt, som der hentydes til, kan bestå af flere elementer. I dette projekt er det funktionen, ’GUILayout.BeginArea’, som opretter et område, der kan redigeres. Det bliver eksempelvis brugt i scriptet, ’LevelSelect’, hvor det enkelte områder, som bliver oprettet indeholder en række knapper. Som ses på billedet, findes der 3 grupper i midten af grænsefladen, som hvert består af et ’BeginArea’. Hver enkelt har skærmkoordinater, som fortæller, hvor på skærmen, knappen skal sidde. Disse koordinator kan dermed gøres dynamiske, og derved opnå en større ensformighed i brugergrænsefladen og mere generisk kode.

Kigger man på koden bag en af disse områder, er det muligt at se mekanikken bag sammenhængen af område og knapper. Koden for et enkelt af disse områder ses her:

GUILayout.BeginArea(new Rect(Screen.width/4-Screen.width/8, Screen.height/2-80, Screen.width, 240));

GUILayout.BeginVertical("");

GUILayout.BeginHorizontal("");

GUILayout.Button("1-1", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));

GUILayout.Button("1-2", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));

GUILayout.Button("1-3", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/15));

GUILayout.EndHorizontal();

GUILayout.Button("Text!", GUILayout.Height(Screen.height/10), GUILayout.Width(Screen.width/5+5));

GUILayout.EndVertical();

GUILayout.EndArea();

Der startes med at oprette et område, som bliver tildelt dynamiske variabler, for at sikre det dynamiske layout. Derefter bliver der påbegyndt et funktion, GUILayout.BeginVertical, som starter en lodret række af elementer. Inde i den påbegyndes den tilsvarende vandrette funktion, som indkapsler 3 knapper, som nu arver ’BeginArea’’s koordinater og automatisk er sat i række via ’BeginVertical’ og ’BeginHorizontal’. Den vandrette række bliver afsluttet med ’EndHorizontal’, hvor efter endnu en knap oprettes, som nu ligger under den første vandrette række. Sidst afsluttes den lodrette række, hvorefter området afsluttes.

Der er altså blevet oprettet en række scener med tilhørende GUI, udviklet i Unity3D’s eget system. Systemet tager højde for skærmens størrelse og dimensioner, og det må dermed siges at opgaven er afsluttet.