Technologia i rozwiązania

Unity

Przepisy na interfejs gry

Francesco Sapio

Helion

[PACKT] PUBLISHING open source community experience distilled

Tytuł oryginału: Unity UI Cookbook

Tłumaczenie: Zbigniew Waśko

ISBN: 978-83-283-2885-3

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/unipig
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 1

Podstawy interfejsu użytkownika

Rozdział ten zawiera następujące przepisy:

- Tworzenie obramowanego sprajta z dwuwymiarowej tekstury.
- Dopasowywanie interfejsu do wymiarów i rozdzielczości ekranu.
- Wstawianie obrazu do interfejsu.
- Nakładanie okrągłej maski na obraz.
- Implementacja przewijania obrazu.
- Implementacja przewijania tekstu za pomocą pionowego suwaka.
- Uaktywnianie przycisków za pośrednictwem klawiatury.
- Porządkowanie interfejsu.

Wprowadzenie

W tym rozdziale postaram się objaśnić, jak należy używać podstawowych narzędzi do tworzenia złożonych interfejsów. Omówię wstawianie i skalowanie elementów interfejsu, stosowanie masek w celu ukształtowania obrazów oraz mechanizm przewijania obrazów i tekstów za pomocą suwaków.

Rozpoczniemy od utworzenia sprajta z obramowaniem. Będzie go można powiększać i pomniejszać bez wprowadzania zniekształceń. Następnie objaśnię, jak zrobić interfejs, który można bez problemu dostosować do rozdzielczości ekranu.

Później przyjrzymy się najpopularniejszemu składnikowi interfejsu, noszącemu nazwę *Image (Script)* (skrypt obrazu). Przy okazji pokażę, jak można zaimplementować bardziej złożone przekształcenia obrazu, np. maskowanie i przewijanie.

Pokażę też, jak stworzyć mechanizm skryptowy umożliwiający aktywowanie przycisków za pośrednictwem klawiatury.

Na koniec przyjrzymy się komponentom organizacyjnym, które pozwalają porządkować i systematyzować rozbudowane interfejsy.

Przygotowania

Żeby zrealizować pierwszy przepis, musimy mieć zainstalowany program Unity 5. Zaczynamy od utworzenia nowego projektu. Projekt może mieć charakter dwu- lub trójwymiarowy, w zależności od tego, jaką grę zamierzamy tworzyć. Jeśli wybierzemy opcję dwóch wymiarów, program automatycznie zaimportuje nasze obrazy jako sprajty. Na potrzeby wykonywanych ćwiczeń wyszukaj kilka ładnych obrazów.

Tworzenie obramowanego sprajta z dwuwymiarowej tekstury

W celu dostosowania dwuwymiarowej tekstury do rozdzielczości ekranu może być konieczne wydzielenie w niej specjalnego obramowania. Pozwoli to wymusić właściwą rozdzielczość bez skalowania tych części obrazu, które mogłyby przez to zmienić swój pierwotny wygląd. Obramowania są często stosowane w przypadku przycisków, których wygląd zależy w dużym stopniu od kształtu narożników i dlatego te ostatnie najczęściej są wyłączane ze skalowania.

Jak to się robi…

1. Najpierw musimy zaimportować obraz. Można to zrobić przez kliknięcie prawym przyciskiem myszy w obrębie panelu *Project* (projekt) i wybranie polecenia *Import New Asset* (zaimportuj nowy element).
2. Za pomocą systemowej przeglądarki plików odszukujemy przygotowany wcześniej obraz i importujemy go. Odtąd będzie on częścią projektu i będziemy mogli go dowolnie używać.

> Obrazy można dodawać do projektu także przez przeciąganie ich bezpośrednio do panelu *Project*.

3. Następnie należy zaznaczyć obraz w panelu *Project*, aby wyświetlić ustawienia jego importu w panelu *Inspector* (inspektor).

4. Aby obraz mógł być użyty jako część interfejsu, właściwość *Texture Type* (typ tekstury) musi być ustawiona na *Sprite (2D and UI)* (sprajt 2D i interfejs). Ustawienie to można zmienić w panelu *Inspector* po zaznaczeniu obrazu, ale jeśli projekt został zadeklarowany jako dwuwymiarowy, program zrobi to za nas i od razu zaimportuje obraz we właściwym trybie.
5. Po sprawdzeniu, czy typ tekstury jest prawidłowy, klikamy przycisk *Sprite Editor* (edytor sprajtów).
6. Na każdym z czterech boków obrazu znajdziemy zielony znacznik. Przez przeciąganie tych znaczników możemy podzielić obraz na części i stworzyć tym samym swoiste obramowanie. Przeciągnij więc te znaczniki ku środkowi obrazu, aby to obramowanie stało się widoczne. Podczas tej czynności należy zwracać uwagę na krzywizny narożników, z którymi możemy mieć do czynienia przy opracowywaniu przycisków. Wszystkie zielone linie oddzielające część środkową od obramowania powinny przebiegać poza krzywiznami. W przeciwnym wypadku krzywizny te zostaną przeskalowane w niewłaściwy dla nich sposób.
7. Na koniec klikamy w prawym górnym rogu okna *Sprite Editor* przycisk *Apply* (zastosuj), aby zatwierdzić wprowadzone zmiany. Okno *Sprite Editor* nie będzie już potrzebne, więc można je zamknąć.
8. Obraz ma już wytyczone obramowanie, a zatem może być należycie skalowany. Aby został prawidłowo umieszczony w interfejsie, musimy go wstawić jako komponent *Image (Script)* typu *Sliced* (pocięty).

Jak to działa…

Czasami niektóre składniki interfejsu wymagają przeskalowania. Nie wszystkie jednak nadają się do skalowania w sposób dowolny, gdyż mogą zyskać wygląd odmienny od tego, o jaki nam chodziło. Dlatego stosujemy obrazy z obramowaniami, które jednoznacznie określają, co i jak powinno być skalowane. Gdy przesuwamy zielone znaczniki w oknie *Sprite Editor*, dokonujemy podziału obrazu na dziewięć części. Każda z nich jest skalowana w inny sposób. Część środkowa jest skalowana zarówno w kierunku *x*, jak i w kierunku *y* (czyli w poziomie i w pionie). Części boczne są skalowane tylko w jednym kierunku — lewa i prawa w pionie, a górna i dolna w poziomie. Części narożne są w ogóle wyłączone ze skalowania. Wszystko to ilustruje poniższy rysunek.

Aby lepiej zrozumieć, dlaczego wyznaczanie obramowania jest tak ważne, rozważmy prosty przykład. Załóżmy, że umieściliśmy w interfejsie przycisk podobny do pokazanego na powyższym rysunku. Jeśli gra będzie uruchamiana na urządzeniach z wyświetlaczami o różnych rozdzielczościach, Unity odpowiednio przeskaluje jej interfejs (więcej informacji na ten temat znajdziesz w następnym przepisie). Oczywiście zależy nam na tym, aby za każdym razem przycisk wyglądał tak, jak go zaprojektowaliśmy. Nie może on zmienić krzywizn narożników, a to oznacza, że one w ogóle nie powinny być skalowane. Informację taką możemy przekazać programowi właśnie przez wyznaczenie odpowiedniego obramowania. Poszczególne części przycisku będą dostosowywane do bieżącej rozdzielczości ekranu, ale dzięki zabezpieczeniu narożników ogólny kształt zostanie zachowany.

Opcje dodatkowe

Poniżej przedstawiam sposób pocięcia sprajta na mniej niż dziewięć części.

Cięcie na mniej niż dziewięć części

Jak już wiemy, w celu ustalenia metody skalowania obrazu można go podzielić na dziewięć części i wytyczyć w ten sposób odpowiednie obramowanie. Nie musimy jednak zawsze dokonywać takiego pełnego podziału. Czasami wystarczy podział uproszczony z niepełnym obramowaniem. Rozwiązanie takie może być przydatne, gdy jakiś element interfejsu przylega do krawędzi ekranu. Obraz będzie wówczas skalowany tylko w tych obszarach, które wyodrębnimy. W takiej sytuacji nie przesuwamy wszystkich zielonych znaczników — jak robiliśmy to w punktach 5. i 6. powyższego przepisu — lecz tylko niektóre z nich. W ten sposób możemy ustalać różne metody skalowania interfejsu w zależności od tego, jaki chcemy mu nadać wygląd w określonej rozdzielczości. Możemy utworzyć np. coś takiego jak na poniższym rysunku.

Patrz także…

- Odpowiednie obramowanie obrazów jest bardzo ważne, jeśli interfejs ma być skalowany w celu dopasowania go do różnych rozdzielczości i platform systemowych, więc może warto przyjrzeć się dokładniej również samemu skalowaniu. Następny przepis, „Dostosowanie rozmiarów interfejsu do wymiarów ekranu i jego rozdzielczości", zawiera omówienie takiego procesu.
- Warto też dowiedzieć się czegoś więcej na temat umieszczania obrazów w interfejsie — w tym celu zapoznaj się z przepisem „Wstawianie obrazów do interfejsu".

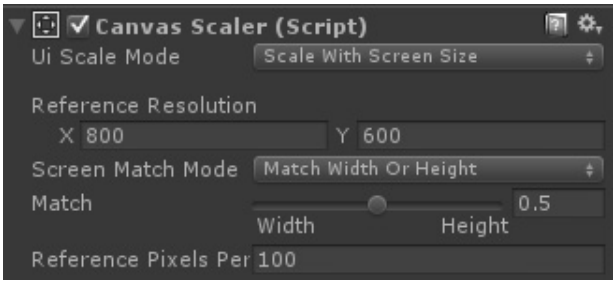
Dostosowanie rozmiarów interfejsu do wymiarów ekranu i jego rozdzielczości

W starszych wersjach programu Unity jedną z najtrudniejszych rzeczy do wykonania było skalowanie interfejsu użytkownika (zwanego wówczas graficznym interfejsem użytkownika, w skrócie GUI). Opanowanie tamtego systemu interfejsowego było dość trudne, a większość funkcji, włącznie ze skalowaniem, należało implementować za pomocą skryptów. W Unity 5 jest to już dużo łatwiejsze, gdyż wystarczy odpowiednio skonfigurować składnik *Canvas Scaler (Script)* (skrypt skalujący płótno), a ten zajmie się skalowaniem wszystkich elementów interfejsu umieszczonych na płótnie (obiekcie *Canvas*).

Jak to się robi…

1. W poprzednim przepisie zobaczyliśmy, że program sam dodał do naszego projektu obiekt *Canvas* — widać go było w panelu *Hierarchy* (hierarchia). Dzieje się tak zawsze, gdy tworzymy jakikolwiek element interfejsu, a wspomniany obiekt jeszcze nie istnieje. Oczywiście możemy też zrobić to samodzielnie — przez kliknięcie panelu *Hierarchy* prawym przyciskiem myszy i wybranie polecenia *UI/Canvas* (interfejs użytkownika/płótno).
2. Jeśli zaznaczymy utworzone płótno, wszystkie jego parametry oraz umieszczone na nim składniki interfejsu zostaną wyświetlone w panelu *Inspector*. W chwili tworzenia płótna automatycznie dołączany jest do niego komponent *Canvas Scaler (Script)*, ale jako że jest to komponent usuwalny, może się zdarzyć, że go nie zobaczymy, a wtedy należy go ponownie dodać. W tym celu klikamy w panelu *Inspector* przycisk *Add Component* (dodaj komponent) i wybieramy opcję *Layout/Canvas Scaler* (układ/skalowanie płótna).

3. Następnie z listy rozwijanej *Ui Scale Mode* (tryb skalowania interfejsu) wybieramy opcję *Scale With Screen Size* (skalowanie zgodne z rozmiarem ekranu) i upewniamy się, że właściwość *Screen Match Mode* (sposób dopasowania do ekranu) jest ustawiona na *Match Width Or Height* (dopasowanie do szerokości lub wysokości). Za pomocą parametru *Match* (dopasowanie) możemy ustalić, czy skalowanie ma uwzględniać bardziej szerokość, czy wysokość. Poniższy rysunek przedstawia sytuację, w której wartość tego parametru wynosi dokładnie `0.5`.



4. W rezultacie wszystkie elementy interfejsu umieszczone na płótnie będą skalowane zgodnie z rozdzielczością ekranu, a więc ich rozmiary będą się automatycznie zmieniać w zależności od urządzenia, na którym gra będzie uruchamiana.

Jak to działa…

W systemie interfejsowym oferowanym przez Unity płótno (*Canvas*) pełni funkcję szczególną. Jest swego rodzaju kontenerem dla wszystkich elementów interfejsu. Elementy leżące poza nim nie będą renderowane. Za każdym razem, gdy jest tworzone, otrzymuje trzy domyślne komponenty, a jednym z nich jest *Canvas Scaler (Script)*, który steruje skalowaniem elementów interfejsu umieszczonych na danym płótnie. Przez manipulowanie jego parametrami możemy ustawić taki sposób skalowania, który będzie najlepiej odpowiadał naszym potrzebom. W szczególności za pomocą parametru *Match* możemy wymusić skalowanie proporcjonalne względem szerokości i wysokości ekranu, na którym gra ma być wyświetlana.

Patrz także…

- Istnieje wiele innych sposobów skalowania elementów interfejsu zgodnie z wymiarami ekranu. Jeśli chciałbyś się z nimi zapoznać, sięgnij po książkę Simona Jacksona *Unity 3D UI Essentials*, wydaną przez Packt Publishing, ISBN 139781783553617 (*https://www.packtpub.com/game-development/unity3d-ui-essentials*).
- Możesz również poczytać o tym komponencie w oficjalnej dokumentacji programu Unity, znajdującej się pod adresem *http://docs.unity3d.com/Manual/script-CanvasScaler.html*.

Wstawianie obrazów do interfejsu

Interfejsy użytkownika zazwyczaj zawierają dużo brązów, więc warto nauczyć się manipulowania nimi za pośrednictwem komponentów *Image (Script)*. Rzeczywiście, komponentów tych będziemy używać we wszystkich przepisach zamieszczonych w tej książce. Na razie skupimy się na wstawianiu i właściwym umieszczaniu obrazu w interfejsie.

Jak to się robi…

1. Zacznij od zaimportowania obrazu do naszego projektu, podobnie jak zrobiliśmy to w pierwszym przepisie z tego rozdziału. Możesz w tym celu kliknąć prawym przyciskiem myszy w panelu *Project* i wybrać z podręcznego menu polecenie *Import New Asset* albo po prostu przeciągnąć obraz na wspomniany panel (wtedy należy pominąć następny etap).
2. W przeglądarce plików odszukaj i wskaż obraz, który ma być importowany.
3. W panelu *Project* zaznacz obraz, a wszystkie jego właściwości staną się dostępne w panelu *Inspector*.
4. Aby wykorzystać zaimportowany obraz w interfejsie jako *Source Image* (obraz źródłowy), należy właściwość *Texture Type* ustawić na *Sprite (2D and UI)*. Jak już widzieliśmy w przepisie pierwszym, jeśli projekt został zadeklarowany jako dwuwymiarowy, wszystkie obrazy są importowane domyślnie jako sprajty.
5. Panel *Inspector* zawiera rozmaite ustawienia importu. Na potrzeby tego przepisu nie musimy zmieniać ich wszystkich, ale warto je bliżej poznać, żeby móc prawidłowo importować obrazy. Wśród tych ustawień są m.in. takie, które określają format, wymiary i metodę kompresji obrazu. Mogą być przydatne np. wtedy, gdy dla poprawienia wydajności gry chcielibyśmy zmniejszyć rozmiar pliku z zapisanym obrazem. O innych ustawieniach będzie jeszcze mowa w punkcie „Opcje dodatkowe”. Po wprowadzeniu stosownych zmian zatwierdź je przez kliknięcie przycisku *Apply* (zastosuj). Jeśli zmiany mają dotyczyć wielu różnych obrazów, proces zatwierdzania może potrwać nawet kilka minut.
6. Następnie umieść obraz w interfejsie. W tym celu kliknij prawym przyciskiem myszy w obrębie panelu *Hierarchy* i wybierz polecenie *UI/Image* (interfejs użytkownika/obraz). Sprawdź też, czy obraz zostanie przypisany do obiektu *Canvas*.

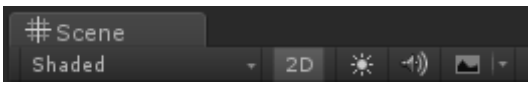
Składniki interfejsu pozostawione poza płótnem nie będą renderowane przez program, więc zawsze sprawdzaj, czy wszystkie są powiązane z obiektem płótna!

Jeśli jakiś składnik nie spełnia tego warunku, po prostu przeciągnij go na obiekt *Canvas*, aby utworzyć stosowne powiązanie.

7. Zaimportowany obraz przeciągnij na pole *Source Image* będące częścią komponentu *Image (Script)*, który z kolei znajduje się w panelu *Inspector*.

> Jeśli obraz ma wytyczone obramowanie do zróżnicowanego skalowania, ustaw jego typ (*Image Type*) na *Sliced* (pocięty).

8. W rezultacie nasz obraz pojawi się w interfejsie, ale przejmie wymiary i współczynniki skalowania od obiektu *Image* (obraz). Jeśli konieczne jest utrzymanie jego pierwotnych wymiarów, można kliknąć przycisk *Set Native Size* (ustaw rozmiar pierwotny).
9. Aby ułatwić sobie pracę nad interfejsem, ustaw dwuwymiarowy widok sceny, jak pokazano na poniższym rysunku, a następnie oddal widok i przeciągnij go tak, żeby całe płótno było widoczne. Na razie jest bardzo duże, więc bądź przygotowany na dość duże oddalenie widoku. W celu skierowania na nie kamery możesz użyć klawisza *F* lub możesz po prostu kliknąć dwukrotnie pozycję *Canvas* w panelu *Hierarchy*.



Do skalowania i umiejscawiania obrazu używamy ostatniego z narzędzi transformujących umieszczonych w lewym górny rogu okna programu (patrz rysunek poniżej). Nosi ono nazwę *Rect* (prostokąt) i można je szybko uaktywnić przez wciśnięcie klawisza *T*. Podczas gdy w trzech wymiarach do przesuwania, obracania i skalowania obiektów potrzebne są odrębne narzędzia, ponieważ mamy tam do czynienia z wieloma stopniami swobody, w dwóch wymiarach możemy te wszystkie operacje wykonać przy użyciu jednego narzędzia.

10. Po zaznaczeniu obrazu w jego narożnikach pojawią się niebieskie punkty. Są to punkty kontrolne narzędzia *Rect*.
11. Przez przeciąganie jednego z nich możemy skalować obraz.

> Aby zachować oryginalne proporcje obrazu, podczas przeciągania niebieskiego punktu przytrzymaj wciśnięty klawisz *Shift*.
>
> Jeśli zamiast *Shift* przytrzymasz *Alt*, uzyskasz możliwość skalowania symetrycznie w dwóch przeciwnych kierunkach.
>
> Jeszcze inny sposób zmieniania wymiarów obrazu polega na przeciąganiu boków prostokąta zamiast narożników.

12. Obraz możemy także swobodnie przemieszczać — wystarczy kliknąć gdziekolwiek wewnątrz prostokąta i przeciągnąć go w odpowiednie miejsce. Należy jednak przy tym pamiętać, że gracz zobaczy tylko te elementy interfejsu, które będą w obrębie płótna.

Jak to działa…

Nowy system interfejsowy w Unity umożliwia dodawanie obrazów w bardzo prosty sposób. Komponent *Image (Script)* pobiera obraz źródłowy (*Source Image*) i rysuje go na ekranie. Dzieje się tak jednak tylko wtedy, gdy obiekt, z którym powiązany jest ten komponent, znajduje się w obrębie płótna. Unity wywołuje funkcje rysowania wyłącznie dla tych obiektów, które są na płótnie.

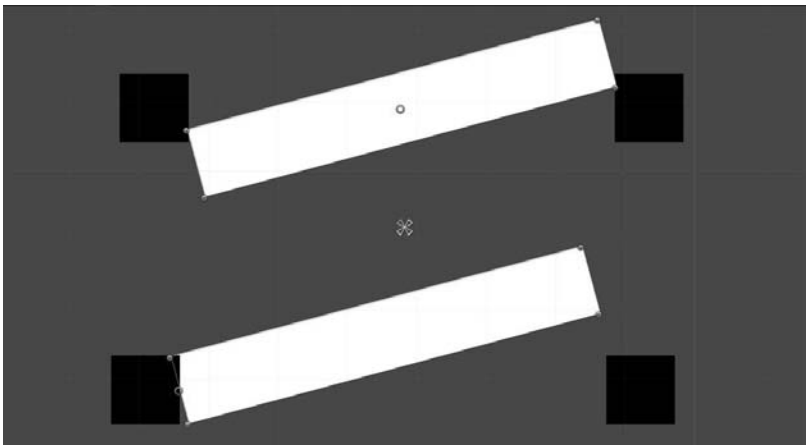
W powyższym przykładzie użyliśmy narzędzia *Rect* do przesuwania i skalowania obrazu, ponieważ w przyjętym przez nas układzie interfejsu wszystkie elementy są reprezentowane przez prostokąty.

Opcje dodatkowe

W projektowaniu interfejsu użytkownika duże znaczenie ma możliwość precyzyjnego obracania obrazu. Ważna jest również możliwość przemieszczania środka obrotu. Przyjrzyjmy się zatem narzędziom, które do tego służą, i zobaczmy, jak należy się nimi posługiwać.

Obracanie obrazu i przesuwanie środka obrotu

Obraz można obrócić przez ustawienie kursora w niewielkiej odległości poza jednym z narożników prostokąta i przeciągnięcie myszą w odpowiednim kierunku. Przeciągnięcie należy wykonać wtedy, gdy kursor przyjmie postać dwóch zakrzywionych strzałek. Położenie środka obrotu możemy dowolnie zmieniać — wystarczy kliknąć znajdujący się w centrum prostokąta niebieski okrąg i przeciągnąć go. Dzięki temu możemy obracać obrazy na różne sposoby. Aby to lepiej zrozumieć, przeanalizuj poniższy rysunek, na którym przedstawiono obrót tego samego prostokąta wokół dwóch różnych środków.



Czarne kwadraty pełnią funkcję układu odniesienia, którego obecność pozwala dostrzec różnicę w skutkach obrotu białego prostokąta. W obu przypadkach prostokąt został obrócony o ten sam kąt. W przykładzie górnym środek obrotu (niebieski okrąg) znajduje się dokładnie w centrum prostokąta, a w przykładzie dolnym został przesunięty na środek lewego boku.

Zmienianie położenia środka obrotu może dawać interesujące efekty, zwłaszcza gdy tworzymy interfejs animowany lub sterowany za pośrednictwem skryptów. Można np. poeksperymentować w ten sposób z interfejsami opisanymi w przepisach „Tworzenie rozszerzalnego elementu z końcowym efektem zanikania" oraz „Tworzenie elementu rozszerzalnego i obrotowego z końcowym efektem zanikania", zamieszczonych w rozdziale 5., „Ozdabianie interfejsu".

Maskowanie obrazu

Często spotykamy w grach elementy interfejsu, które wcale nie są prostokątne. Najprostszą metodą uzyskiwania rozmaitych kształtów jest maskowanie. Jednym z często stosowanych kształtów jest koło, więc zajmiemy się teraz tworzeniem maski o takim właśnie kształcie. Może się to przydać do konstruowania okrągłych przycisków lub okrągłych teł dla ikon. Ważnym komponentem będzie teraz dla nas komponent o nazwie *Mask* (maska).

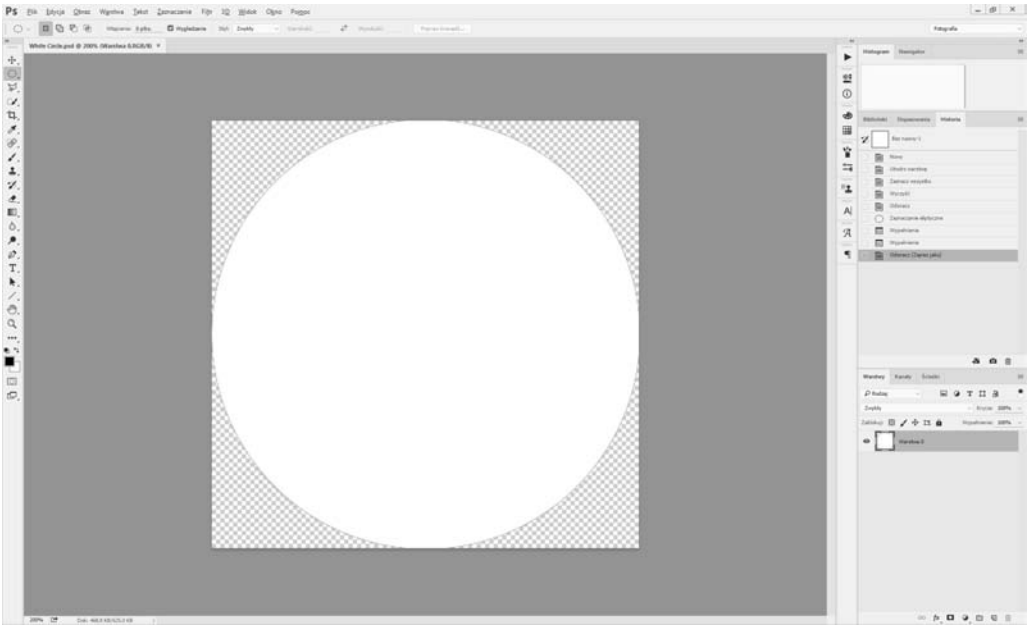
Jak to się robi…

1. Przede wszystkim musimy utworzyć maskę, w tym przypadku okrągłą. Maska sama w sobie jest obrazem, więc do jej tworzenia musimy wykorzystać program graficzny. Zaczynamy od wygenerowania nowego płótna o takiej samej liczbie pikseli wzdłuż szerokości i wysokości, czyli kwadratowego.
2. Czynności z tego etapu oraz z następnego będą zależeć od tego, z jakiego programu graficznego korzystamy. Po prostu musimy narysować białe koło na środku płótna.
3. Następnie musimy spowodować, aby tło było przezroczyste. Ostateczny efekt powinien być podobny do tego, który widać na rysunku na następnej stronie (w tym przypadku użyto programu Photoshop).

Jeśli używasz Photoshopa, możesz taką maskę łatwo utworzyć za pomocą narzędzia do rysowania elips. Po prostu przeciągnij myszą wzdłuż przekątnej kwadratu, trzymając przy tym wciśnięty klawisz *Shift*.

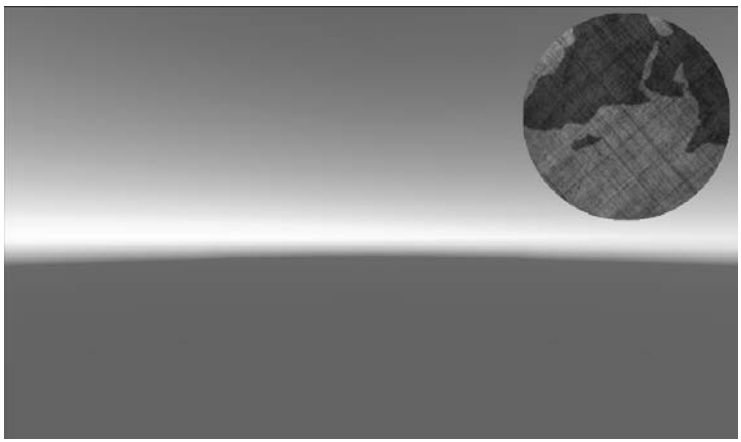
Jeżeli nie masz żadnego programu graficznego lub nie chcesz tracić czasu na jego uruchamianie i rysowanie czegokolwiek, możesz wykorzystać obraz zamieszczony w materiałach powiązanych z tą książką.

4. Gotową maskę importujemy do projektu w Unity. W tym celu klikamy prawym przyciskiem myszy w obrębie panelu *Project* i wybieramy polecenie *Import New Asset*.



5. Unity poprosi nas o wskazanie obrazu, który chcemy zaimportować, więc odszukujemy folder z przygotowaną wcześniej maską i importujemy ją.
6. Aby uzyskać dostęp do właściwości importowanej maski, wyświetlanych w panelu *Inspector*, zaznaczamy ją w panelu *Project*.
7. Jeśli projekt nie został zadeklarowany jako dwuwymiarowy, ustawiamy właściwość *Texture Type* na *Sprite (2D and UI)* i klikamy przycisk *Apply*.
8. Teraz wstawiamy do interfejsu nowy panel. Można to zrobić przez kliknięcie prawym przyciskiem myszy w obszarze panelu *Hierarchy* i wybranie polecenia *UI/Panel* (interfejs/panel). Oczywiście upewniamy się, że ów panel jest przypisany do obiektu *Canvas*.
9. W komponencie *Image (Script)* ustawiamy naszą maskę jako *Source Image*.
10. Aby uzyskać kształt dokładnie taki, jaki utworzyliśmy w programie graficznym, musimy przywrócić obrazowi jego pierwotne proporcje. W tym celu klikamy przycisk *Set Native Size* i dopiero potem, jeśli zajdzie taka potrzeba, stosujemy skalowanie proporcjonalne (przy wciśniętym klawiszu *Shift*).
11. Kolejna czynność, jaką musimy wykonać, to przekształcenie panelu w maskę. Dodajemy więc do naszego panelu komponent *Mask (Script)* (skrypt maskowy).
12. Teraz musimy przygotować obraz, na który nałożymy maskę. Tworzymy zatem nowy obiekt obrazowy w ramach panelu — w panelu *Hierarchy* klikamy prawym przyciskiem myszy i wybieramy opcję *UI/Image*.

13. Komponent *Image (Script)* powstałego obiektu obrazowego wymaga wskazania konkretnego obrazu. W tym przypadku wybieramy ten, który zamierzamy maskować, i po prostu przeciągamy go na pole *Source Image*. Odtąd obraz ten będzie widoczny tylko wewnątrz okrągłej maski.
14. W razie potrzeby można kliknąć przycisk *Set Native Size* i zastosować skalowanie proporcjonalne.
15. Na zamieszczonym niżej rysunku pokazano rezultat zamaskowania w ten sposób obrazu przedstawiającego starodawną mapę.



Jak to działa…

Jak widać, Unity maskuje jeden obraz za pomocą drugiego. Dlatego komponent *Mask (Script)* dołączamy do obiektu posiadającego komponent *Image (Script)*. Dzięki temu, że maska jest obrazem, możemy jej nadać taki kształt, jaki tylko zechcemy. Musimy jedynie pamiętać, że obraz maskowany będzie widoczny wyłącznie w tych obszarach, w których maska jest biała.

Patrz także…

- Jeśli chcesz dowiedzieć się więcej o komponencie *Mask (Script)*, zajrzyj do dokumentacji programu Unity (*http://docs.unity3d.com/Manual/script-Mask.html*).

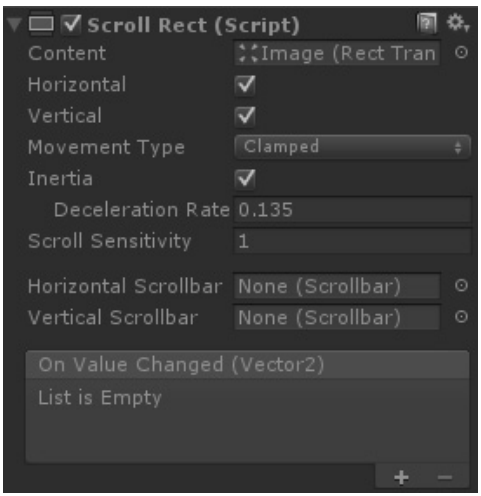
Przewijanie obrazu

Akcja wielu gier toczy się w olbrzymich światach, najczęściej fantastycznych. Mapy takich światów też są duże i żeby gracz mógł dostrzec ich szczegóły, musi mieć możliwość przewijania obszarów widzianych w odpowiednim powiększeniu.

W tym przepisie pokażę, jak coś takiego wykonać za pomocą dwóch komponentów: poznanego już *Mask (Script)* oraz nieomawianego jeszcze *Scroll Rect (Script)* (skrypt przewijania prostokąta). Ten drugi będzie odpowiadał za całą logikę mechanizmu przewijania elementów interfejsu.

Jak to się robi…

1. Żeby nie zaczynać wszystkiego od nowa, wykorzystamy maskę utworzoną w ramach poprzedniego przepisu. Oczywiście wprowadzimy pewne zmiany, aby efekt przewijania był dobrze widoczny.
2. Przede wszystkim musimy wyposażyć panel maskujący w nowy komponent o nazwie *Scroll Rect (Script)*.
3. Zachowując zaznaczenie panelu, przeciągamy przypisany do niego obraz na pole *Content* (zawartość) w komponencie *Scroll Rect (Script)*.
4. Sposób przewijania jest domyślnie ustawiony jako elastyczny (*Elastic*) i w większości przypadków ustawienie to sprawdza się bardzo dobrze, lecz ze względu na okrągły kształt maski byłoby lepiej, gdyby obraz nie wychodził poza jej obręb. Mogłoby to zepsuć ogólne wrażenie estetyczne i raczej byłoby źle przyjęte przez gracza. Dlatego z listy rozwijanej *Movement Type* (rodzaj ruchu) wybierzemy opcję *Clamped* (z ograniczeniem).
5. Dodatkowe szczegóły przewijania można ustalić przez dobór wartości parametrów *Deceleration Rate* (siła hamowania) i *Scroll Sensitivity* (czułość przewijania). Warto trochę poeksperymentować, żeby wybrać wartości optymalne.
6. Ostatecznie komponent *Scroll Rect (Script)* powinien wyglądać mniej więcej tak, jak pokazano na poniższym rysunku.



Jak to działa…

Przede wszystkim trzeba pamiętać, że wykorzystaliśmy obraz, który już wcześniej został zamaskowany. Tak też robi się zazwyczaj z obrazami, które mają być przewijane. Maskowanie pozwala graczowi skupić się na wybranym fragmencie obrazu lub tekstu, a poza tym umożliwia zmieszczenie nawet bardzo dużego dokumentu na stosunkowo niewielkim fragmencie ekranu. A zatem skoro mieliśmy już zamaskowany obraz, wystarczyło dodać do obiektu nadrzędnego komponent *Scroll Rect (Script)*, który wykonuje całą pracę związaną z obsługą przewijania i można go skonfigurować za pośrednictwem kilku parametrów.

Zmieniać możemy np. parametr *Movement Type*. Wybranie opcji *Clamped* uniemożliwi obrazowi wyjście poza obszar wyznaczony przez maskę. Opcja *Elastic* powoduje odbijanie się przewijanej zawartości od krawędzi maski, a opcja *Unrestricted* (nieograniczony) znosi wszelkie ograniczenia ruchu. Parametr *Deceleration Rate* odpowiada za efekt hamowania, ale działa tylko wtedy, gdy zaznaczone jest pole *Inertia* (bezwładność). Od ustawionej tu wartości zależy, jak szybko nastąpi całkowite zatrzymanie przewijanej zawartości. Przy wartości `0` ruch ustaje natychmiast, gdy tylko użytkownik przerwie przewijanie, a przy wartości `1` efekt hamowania w ogóle nie występuje. Ważnym parametrem jest *Scroll Sensitivity*, gdyż od jego wartości zależy reakcja komponentu na zdarzenie przewijania wygenerowane przez użytkownika.

Patrz także…

- Jeśli szukasz więcej informacji, zawsze możesz zajrzeć do dokumentacji programu Unity. Informacje o maskach znajdziesz na stronie *http://docs.unity3d.com/Manual/script-Mask.html*.
- A jeśli chcesz dowiedzieć się czegoś więcej o komponencie *Scroll Rect (Script)*, przestudiuj zawartość strony *http://docs.unity3d.com/Manual/script-ScrollRect.html*.

Przewijanie tekstu za pomocą pionowego suwaka

Czasami w grach występują dość długie teksty, których bez przewijania nie dałoby się przeczytać. Jednym z rozwiązań tego problemu jest wyświetlanie takiego tekstu w oknie wyposażonym w suwak, za pomocą którego można tekst szybko i wygodnie przewijać; dodatkowo suwak ten pozwala się zorientować, która część całości jest obecnie wyświetlana. Aby coś takiego zrobić, postąpimy podobnie jak w poprzednim przepisie, tyle że miejsce obrazu zajmie tekst i wprowadzimy dodatkowy element w postaci pionowego paska do przewijania.

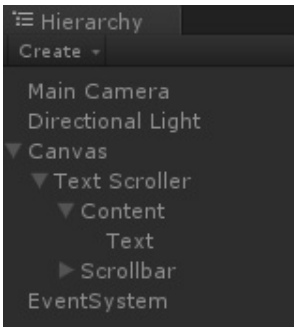
Po wykonaniu wszystkich czynności będziemy w stanie przewijać tekst za pomocą komponentów *Scroll Rect (Script)* oraz *Scrollbar (Script)* (skrypt paska do przewijania).

Jak to się robi…

1. Rozpoczynamy od utworzenia nowego panelu w ramach naszego interfejsu — klikamy prawym przyciskiem myszy w panelu *Hierarchy* i wybieramy opcję *UI/Panel*.
2. Następnie zmieniamy rozmiar tego panelu, aby uzyskać widok sceny podobny do pokazanego na poniższym rysunku.

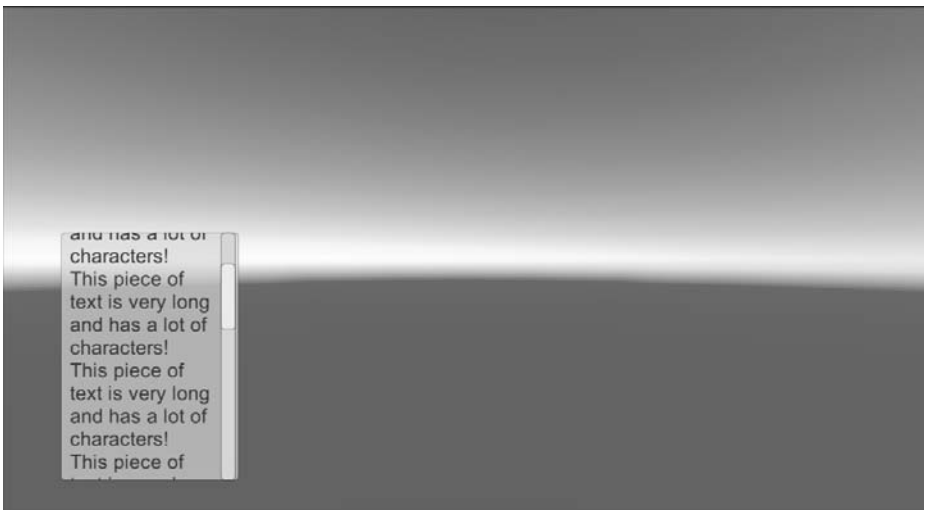
3. Dobrą praktyką jest utrzymywanie należytego porządku w elementach projektu. Dlatego od razu nadajemy panelowi czytelną nazwę, np. **`Text Scroller`** (przewijacz tekstu).
4. Tak jak w poprzednich dwóch przepisach, dodajemy komponent *Mask (Script)*. Żeby nie przeszkadzał nam w oglądaniu sceny, możemy go na razie wyłączyć.
5. Następnie klikamy prawym przyciskiem myszy pozycję *Text Scroller* i wybieramy polecenie *Create Empty* (utwórz pusty). W ten sposób tworzymy pusty obiekt podporządkowany panelowi *Text Scroller*. Oczywiście nadajemy mu odpowiednią nazwę, np. **`Text Content`** (zawartość tekstowa).
6. Do tego obiektu dodajemy komponent *Scroll Rect (Script)*. Teraz musimy jeszcze dodać tekst. W tym celu klikamy prawym przyciskiem myszy pozycję *Text Content* i wybieramy *UI/Text* (interfejs/tekst).
7. Tekst możemy wpisać albo skopiować i wkleić w polu *Text* (tekst) widocznym w górnej części komponentu.

8. Za pomocą narzędzia *Rect* zmieniamy rozmiar obszaru z tekstem w taki sposób, aby otrzymać prostokąt węższy od panelu *Text Scroller*, ale za to wyższy od niego. Wysokość zwiększamy tak długo, aż cały tekst zmieści się w prostokącie. Aby mieć jakieś wyobrażenie o tym, jak powinien wyglądać ten prostokąt, spójrz na rysunek zamieszczony po etapie 10.
9. Ponownie klikamy prawym przyciskiem myszy pozycję *Text Scroller* i wybieramy tym razem opcję *UI/Scrollbar* (interfejs/pasek przewijania). Teraz panel *Hierarchy* powinien wyglądać tak, jak pokazano na rysunku poniżej.



10. Następnie w komponencie *Scrollbar (Script)* zmieniamy kierunek paska (właściwość *Direction*) na *Bottom To Top* (od dołu do góry). Dopasowujemy rozmiar paska do rozmiaru wolnej przestrzeni w panelu i tam też go umieszczamy (jak na poniższym rysunku).

11. Musimy również powiązać ten pasek z tekstem. W tym celu zaznaczamy pozycję *Text Content* i do pola *Vertical Scrollbar* (pionowy pasek przewijania) przeciągamy pozycję *Scrollbar* (pasek przewijania). Wyłączamy także opcję *Horizontal* (poziomo), aby wymusić przewijanie tylko w kierunku pionowym.
12. Na koniec włączamy maskę panelu *Text Scroller* (tę, którą wcześniej wyłączyliśmy, żeby nam nie przeszkadzała) i klikamy przycisk odtwarzania, aby przetestować to, co zrobiliśmy. Jeśli chcesz, możesz, tak jak w poprzednim przepisie, wyłączyć efekt elastyczności — w komponencie *Scroll Rect (Script)* zmień wartość parametru *Movement Type* na *Clamped*.
13. Teraz możemy przewijać tekst w panelu *Text Scroller*. Ostatecznie powinien on wyglądać tak, jak pokazano na poniższym rysunku[1].



Jak to działa…

Cała procedura jest podobna do tej z poprzedniego przepisu, tyle że utworzyliśmy maskę nie dla obrazu, a dla tekstu. Maskę tę tymczasowo ukryliśmy, aby ułatwić sobie umieszczanie tekstu. W roli mechanizmu przewijającego użyliśmy komponentu *Scroll Rect (Script)*. Ostatecznie musieliśmy zmienić wymiary obszaru z tekstem, aby był w całości widoczny na ekranie. Dodaliśmy również pasek do przewijania i powiązaliśmy go z komponentem *Scroll Rect (Script)*. Przez odpowiednie ustawienie opcji tego komponentu i właściwości paska ograniczyliśmy przewijanie tylko do kierunku pionowego. Na koniec przywróciliśmy działanie maski, przez co tekst stał się widoczny wyłącznie w wyznaczonym dla niego obszarze.

[1] W opisie całej procedury pominięto powiązanie obiektu *Text* z obiektem *Text Content*. Bez tego przewijanie nie będzie działało. Aby takie powiązanie utworzyć, należy zaznaczyć obiekt *Text Content* i w jego komponencie *Scroll Rect (Script)* jako wartość parametru *Content* umieścić obiekt *Text — przyp. tłum.*

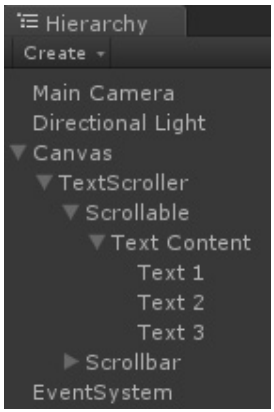
Opcje dodatkowe

Poniżej przedstawiam opisy dwóch innych metod przewijania tekstu, które można wykorzystywać w zależności od konkretnych warunków projektowych.

Przewijanie kilku łamów jednocześnie

Co robić w sytuacji, gdy np. tekst składa się z kilku łamów? Oto zmiany, jakie należy wówczas wprowadzać w powyższym przepisie.

Przede wszystkim tworzymy nowy pusty obiekt i przypisujemy go do panelu *Text Scroller*. Następnie dodajemy do tego obiektu komponent *Scroll Rect (Script)* i przeciągamy utworzony wcześniej obiekt *Text Content* do pola *Content*. Później usuwamy komponent *Scroll Rect (Script)* z obiektu *Text Content*. Na koniec dodajemy do obiektu *Text Content* dowolną liczbę obiektów tekstowych, a zawarty w nich tekst będzie można przewijać. Jeśli dodamy trzy takie obiekty, panel *Hierarchy* będzie wyglądał tak, jak zaprezentowano na poniższym rysunku.



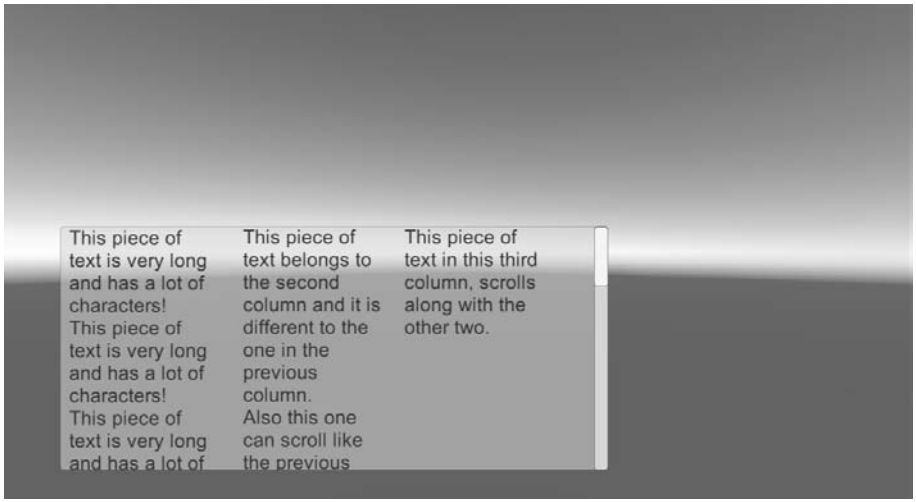
Warto przy tym pamiętać, że w celu właściwego rozmieszczenia tekstu na ekranie dobrze jest wyłączyć tymczasowo maskę przypisaną do panelu *Text Scroller*. Umożliwi to obserwację wszystkich obiektów w całości, a nie tylko w obszarze maski.

Ostatecznie nasza scena powinna wyglądać tak, jak pokazano na rysunku na następnej stronie.

> W celu właściwego dopasowania tekstu do panelu może być konieczna zmiana rozmiarów wszystkich komponentów. Tylko wtedy efekt przewijania będzie naturalny i zgodny z naszymi planami.

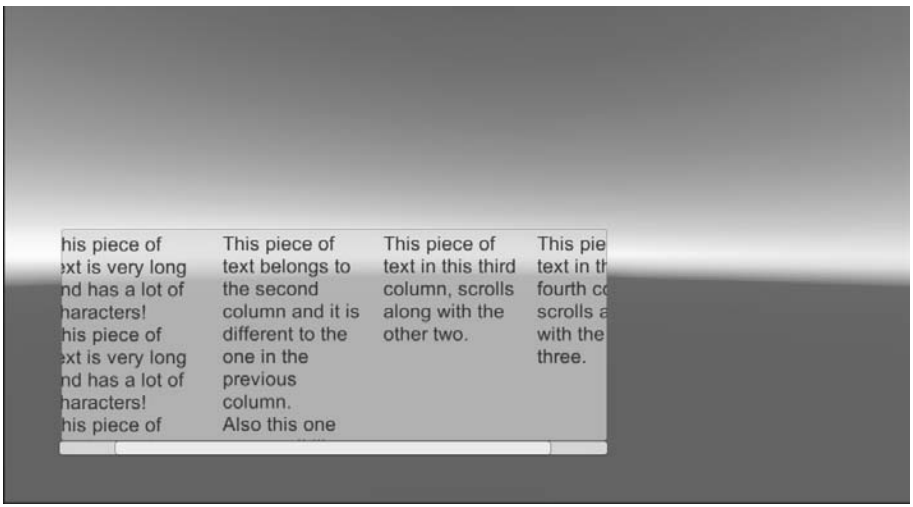
Przewijanie poziome

Gdyby zaszła potrzeba zastosowania przewijania w poziomie, choć jest to raczej rzadko spotykane, możemy po prostu zmienić orientację i inne ustawienia paska, który już istnieje. W grach ten sposób przewijania stosuje się najczęściej przy wyświetlaniu rozmaitych spisów.



Zaczynamy od zmodyfikowania tego, co wykonaliśmy w punkcie 8. Tym razem wysokość obiektu *Text* musi być nieco mniejsza od wysokości panelu *Text Scroller*, a szerokość powinna być na tyle duża, aby pomieścić całość tekstu. Następnie w punkcie 10. zamiast opcji *Bottom To Top* jako kierunek paska wybieramy *Left To Right* (od lewej do prawej). Na koniec w punkcie 11. obiekt *Scrollbar* przeciągamy do pola *Horizontal Scrollbar* (poziomy pasek przewijanie), a nie do pola *Vertical Scrollbar* i wyłączamy opcję *Vertical* (pionowo) zamiast *Horizontal*.

Czytanie tekstu szerszego niż przeznaczony do jego wyświetlania panel jest bardzo niewygodne, a przewijanie poziome jedynie umożliwia odczytanie całości, ale wcale tej czynności nie ułatwia. O wiele łatwiej czyta się taki tekst, jeśli jest podzielony na kilka łamów, co można dość łatwo uzyskać przez utworzenie wewnątrz obiektu *Text Content* odpowiedniej liczby obiektów typu *Text* i ułożenie ich obok siebie jak łamy tekstu. Przykład takiego rozwiązania jest pokazany na poniższym rysunku.



I znów w celu właściwego dopasowania tekstu do panelu może być konieczna zmiana rozmiarów wszystkich komponentów. Tylko wtedy efekt przewijania będzie naturalny i zgodny z naszymi planami.

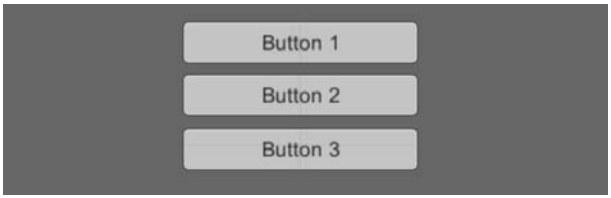
Uaktywnianie przycisków za pośrednictwem klawiatury

W grach bardzo często polecenia menu mają przypisane indywidualne skróty klawiszowe. Jeśli gracz chce szybko włączyć jakąś funkcję, musi mieć taką możliwość. Ogólnie rzecz biorąc, wszędzie tam, gdzie trzeba uaktywnić kilka elementów interfejsu, takich jak przyciski lub przełączniki, powinno być możliwe wykonanie tego przez wciśnięcie odpowiednich klawiszy.

W tym przepisie pokażę, jak można wykryć wciśnięcie klawisza i uaktywnić na tej podstawie określony element interfejsu. Wykonamy to wszystko za pomocą uniwersalnego skryptu, którego będzie można użyć w odniesieniu do dowolnego elementu interfejsu bez potrzeby modyfikowania kodu.

Jak to się robi…

1. Zaczynamy od utworzenia panelu. Klikamy prawym przyciskiem myszy w panelu *Hierarchy* i wybieramy polecenie *UI/Panel*. Od razu możemy mu nadać jakąś sensowną nazwę, żeby potem nie pogubić się w gąszczu rozmaitych obiektów.
2. Następnie tworzymy na tym panelu przycisk. Można to zrobić przez kliknięcie prawym przyciskiem myszy w panelu *Hierarchy* i wybranie polecenia *UI/Button* (interfejs/przycisk).
3. Aby ułatwić sobie rozpoznawanie, który przycisk został wciśnięty, zmienimy niektóre parametry komponentu *Button (Script)* (skrypt przycisku). Możliwa jest np. zmiana koloru dla przycisku w stanie neutralnym (*Normal Color*), wyróżnienia (*Highlighted Color*) i wciśnięcia (*Pressed Color*). Na potrzeby naszego przykładu zmieniamy *Highlighted Color* na czerwony, *Pressed Color* na zielony, a *Normal Color* pozostawiamy biały. Zmieniamy również ustawienie *Navigation* (nawigacja) na *Vertical* (pionowa).
4. Skoro mamy wybierać spośród różnych przycisków, musimy utworzyć ich więcej. W tym celu powielimy ten, który już istnieje. Po prostu wciskamy klawisze *Ctrl+D*, a następnie przesuwamy kopię poniżej oryginału. Nazwę oryginału zmieniamy na **`Button 1`** (przycisk 1), a kopii na **`Button 2`** (przycisk 2). Zmieniamy także napisy na przyciskach, czyli zawartość pól *Text* w komponentach *Text (Script)* (skrypt tekstu) obiektów *Tekst* przypisanych do poszczególnych przycisków. Dzięki temu będzie można rozróżniać przyciski również na ekranie. Na koniec powtarzamy procedurę powielania, aby uzyskać sumie trzy przyciski. Ostatecznie nasza scena powinna wyglądać podobnie jak ta na rysunku na następnej stronie.



5. W końcu nadszedł czas na napisanie pierwszego skryptu. W panelu *Inspector* klikamy przycisk *Add Component* i wybieramy opcję *New Script* (nowy skrypt). Jako nazwę skryptu wpisujemy **`ButtonThroughKeySelection`** (uaktywnianie przycisku za pomocą klawisza) i po upewnieniu się, że jako język skryptu wybrany jest *C Sharp*, klikamy przycisk *Create and Add* (utwórz i dodaj).
6. Następnie klikamy dwukrotnie nazwę skryptu, aby otworzyć go w aplikacji MonoDevelop.
7. Gdy używamy klas związanych z interfejsem, zawsze musimy dodawać na początku skryptu odpowiednią klauzulę `using`. W ten sposób zabezpieczamy się przed błędami kompilacji, jakie mogłyby powstać podczas kompilacji skryptu przez Unity. Musimy również dodać klauzulę informującą, że będziemy się odwoływać do zdarzeń. Dlatego początek naszego skryptu powinien wyglądać następująco:

```
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine;
using System.Collections;
```

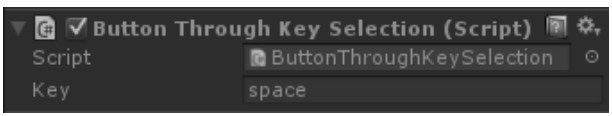
8. Poza tym musimy zadeklarować zmienną dla wciśniętego klawisza. Jeśli to zrobimy, będziemy mogli później ustawiać tę zmienną w panelu *Inspector*.

```
public class ButtonThroughKeySelection: MonoBehaviour {
   public string key;
```

9. Jako że nie musimy ustalać wartości początkowych, możemy usunąć funkcję `Start()`. Aby zaimplementować rozpoznawanie wciśniętego klawisza i uaktywnianie odpowiedniego przycisku, musimy napisać funkcję `Update()`w następujący sposób:

```
public void Update()
   if (Input.GetKeyDown (key))
   {
      EventSystem.current.SetSelectedGameObject(this.gameObject);
   }
}
```

10. Zapisujemy skrypt i dodajemy go do pierwszego przycisku.
11. Następnie wpisujemy nazwę klawisza, którego wciśnięcie ma uaktywniać przycisk, np. **`space`**, jeśli tym klawiszem ma być spacja. Komponent skryptowy powinien wtedy wyglądać tak, jak pokazano na poniższym rysunku.



12. Na koniec klikamy przycisk odtwarzania i sprawdzamy, czy wszystko działa prawidłowo. Po uaktywnieniu pierwszego przycisku przez wciśnięcie klawisza wpisanego jako wartość zmiennej możemy uaktywniać kolejne przyciski, używając klawiszy ze strzałkami. Ponowne wciśnięcie klawisza spacji spowoduje jednak powrót do przycisku pierwszego (tego, do którego ten klawisz został przypisany).

Jak to działa…

Na potrzeby testu utworzyliśmy trzy przyciski. Aby skutki działania skryptu były dobrze widoczne, zmieniliśmy odpowiednie właściwości tych przycisków. Ze względu na ich pionowy układ ustawiliśmy wartość właściwości *Navigation* na *Vertical*.

Pisanie skryptu rozpoczęliśmy od klauzul `using UnityEngine.UI` oraz `using UnityEngine.Event` `↪Systems`. Pierwsza sygnalizuje, że w skrypcie będą występowały odwołania do elementów interfejsu. Klauzuli tej będziemy używać również w wielu innych skryptach prezentowanych w tej książce. Druga klauzula sygnalizuje, że w skrypcie będzie wykorzystywany bezpośrednio system zdarzeniowy (*Event System*).

W głównej części skryptu zadeklarowaliśmy publiczną (`public`) zmienną łańcuchową. Publiczny charakter tej zmiennej oznacza, że jej wartość można określić później w panelu *Inspector*. O tym, jaki klawisz powiązać z danym przyciskiem, nie musimy więc decydować już na etapie pisania skryptu.

W funkcji `Update()` następuje sprawdzenie za pomocą instrukcji `if (Input.GetKeyDown (key)`, czy podany klawisz został wciśnięty. Otóż funkcja `Input.GetKeyDown(string)` zwraca wartość `true` (prawda), jeśli klawisz określony jako `string` jest wciśnięty, a jeśli jest niewciśnięty, wartością zwracaną jest `false` (fałsz). Należy pamiętać, że wartość zmiennej `key` jest ustawiana w panelu *Inspector*, więc można ją swobodnie dopasowywać do projektu gry. Jeżeli chcesz się dowiedzieć więcej na temat wykrywania wciśniętych klawiszy, zajrzyj do punktu „Patrz także…”.

Na koniec, gdy właściwy klawisz został wciśnięty, powinno nastąpić uaktywnienie powiązanego z nim przycisku. Zadanie to wykonuje funkcja o nazwie `EventSystem.current.SetSelectedGame` `↪Object(this.gameObject)`. Początek tej nazwy, `EventSystem.current`, stanowi odwołanie do bieżącego systemu zdarzeniowego. Pozostałą część stanowi funkcja `SetSelectedGameObject` `↪(gameObject)`, której działanie polega na uaktywnieniu obiektu przekazanego jej za pośrednictwem parametru `gameObject`. W tym przypadku użyliśmy konstrukcji `this.gameObject`, aby wskazać na obiekt, do którego skrypt jest przypisany, czyli na interesujący nas przycisk.

Przez sparametryzowanie wszystkich ustawień, włącznie ze zmienną `key`, której wartość można ustalić dla każdej instancji skryptu oddzielnie, jesteśmy w stanie wykorzystać ten skrypt w odniesieniu do wielu przycisków i dopasować go do nich bez konieczności wprowadzania jakichkolwiek zmian w samym kodzie.

Patrz także…

- Podczas pisania skryptu używaliśmy rozmaitych funkcji, z których najważniejszą była `GetKeyDown()`. Za jej pomocą można stwierdzić, czy dany klawisz jest wciśnięty, czy nie. Więcej informacji na jej temat znajdziesz pod adresem: *http://docs.unity3d.com/ScriptReference/Input.GetKeyDown.html*.
- Funkcja ta potrzebuje jednak parametru określającego klawisz, którego wciśnięcie ma być monitorowane. Pełną listę nazw klawiszy znajdziesz pod adresem *http://docs.unity3d.com/ScriptReference/KeyCode.html*.
- W skrypcie użyliśmy także funkcji należącej do systemu zdarzeniowego. Jest to system mocno rozbudowany i często wykorzystywany w grach. Jeśli chcesz dowiedzieć się czegoś więcej na jego temat, zajrzyj na stronę *http://docs.unity3d.com/ScriptReference/EventSystems.EventSystem.html*.

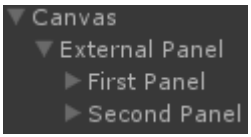
Stosowanie komponentów układu interfejsu

Często zachodzi potrzeba rozmieszczenia dużej liczby obiektów w sposób uporządkowany i symetryczny wewnątrz jakiegoś większego elementu interfejsu. Na szczęście Unity ma wbudowany system automatycznego układania interfejsu (*Auto-Layout*) z możliwością zagnieżdżania jednych elementów w drugich. Strukturę takiego interfejsu możemy kontrolować za pośrednictwem specjalnych komponentów. Umiejętność posługiwania się nimi jest niezbędna, jeśli chce się szybko tworzyć przejrzyste interfejsy, i właśnie jej nabyciu ma służyć realizacja niniejszego przepisu.

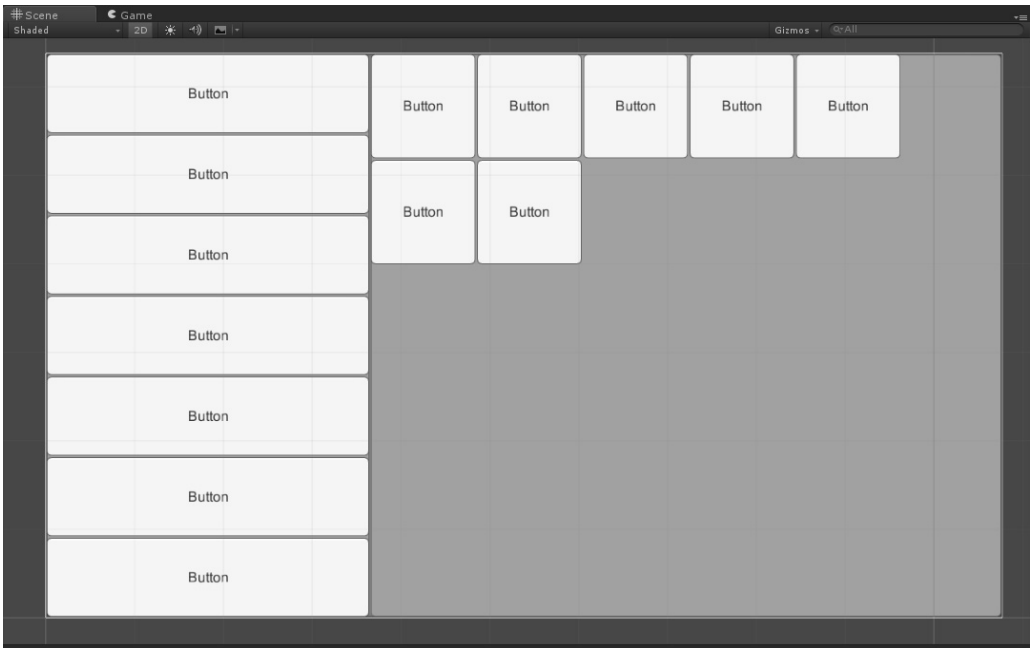
Jak to się robi…

1. Rozpoczynamy od utworzenia panelu, na którym będziemy układać obiekty podrzędne. Klikamy więc prawym przyciskiem myszy w panelu *Hierarchy* i wybieramy opcję *UI/Panel*. Nowemu panelowi nadajemy nazwę **`First Panel`** (panel pierwszy).
2. W utworzonym panelu chcemy umieścić kilka innych elementów. Tym razem będą to przyciski. Tworzymy więc pierwszy z nich, klikając pozycję *First Panel* i wybierając z podręcznego menu polecenie *UI/Button*.
3. Jeden przycisk to za mało, żeby zobaczyć, jak działają komponenty układu. A zatem wciskamy klawisze *Ctrl+D* tyle razy, ile kopii przycisku chcemy utworzyć. Niczego nie porządkujemy ani nie układamy, gdyż to mają zrobić za nas komponenty układu.
4. Ze względu na to, że będziemy chcieli przetestować różne komponenty układu, skopiujemy także cały panel — zaznaczamy jego nazwę i wciskamy klawisze *Ctrl+D*. Nazwę nowego panelu zmieniamy na **`Second Panel`** (panel drugi).

5. Tworzymy też trzeci panel, któremu nadajemy nazwę *External Panel* (panel zewnętrzny), i umieszczamy w nim dwa poprzednie. Ostatecznie struktura naszego interfejsu powinna wyglądać tak, jak pokazano na poniższym rysunku.



6. Następnie dodajemy do panelu zewnętrznego komponent *Horizontal Layout Group (Script)* (skrypt grupy układu poziomego), do panelu pierwszego — komponent *Vertical Layout Group (Script)* (skrypt grupy układu poziomego) i do panelu drugiego — komponent *Grid Layout Group (Script)* (skrypt grupy układu siatkowego). Jak widać, system *Auto-Layout* zrobił, co do niego należało. Aby lepiej zrozumieć sposób jego działania, dodaj jeszcze kilka nowych przycisków lub powiel istniejące i zaobserwuj, jak będzie się zmieniało ich rozmieszczenie. Ostatecznie powinieneś uzyskać układ zbliżony do pokazanego na poniższym rysunku.



Jak to działa…

System *Auto-Layout* składa się z dwojakiego rodzaju elementów: *Layout Elements* (elementy układu) i *Layout Controllers* (sterowniki układu). Do pierwszej grupy należą wszystkie obiekty mające komponent *Rect Transform* (przekształcenia prostokąta). Posiadają one pewną wiedzę na temat swoich wymiarów, ale nie są w stanie tymi wymiarami bezpośrednio sterować. Natomiast sterowniki układu są elementami, które potrafią sterować zarówno wymiarami, jak i położeniem elementów układu. Ich wpływ obejmuje nie tylko te elementy układu, do których są przypisane, ale również elementy podrzędne.

W powyższym przykładzie użyliśmy komponentów *Horizontal Layout Group (Script)*, *Vertical Layout Group (Script)* i *Grid Layout Group (Script)*. Zasady ich działania są bardzo podobne. Każdy z nich zajmuje się układem obiektów podrzędnych w ramach obiektu, do którego został przypisany, i odpowiednio modyfikuje rozkład elementów interfejsu.

Patrz także…

Jeśli chcesz poszerzyć swoją wiedzę na temat sterowników układu, zajrzyj do oficjalnej dokumentacji programu Unity. Oto adresy, pod którymi znajdziesz właściwe zagadnienia:

- **Horizontal Layout Group:**
  http://docs.unity3d.com/Manual/script-HorizontalLayoutGroup.html;
- **Vertical Layout Group:**
  http://docs.unity3d.com/Manual/script-VerticalLayoutGroup.html;
- **Grid Layout Group:**
  http://docs.unity3d.com/Manual/script-GridLayoutGroup.html;
- **Content size fitter** (dostosowywacz rozmiarów zawartości):
  http://docs.unity3d.com/Manual/script-ContentSizeFitter.html;
- **Aspect ratio fitter** (dostosowywacz proporcji):
  http://docs.unity3d.com/Manual/script-AspectRatioFitter.html.

Pobieranie przykładowego kodu

Pliki z przykładowymi kodami są dostępne na serwerach wydawnictwa Helion, znajdujących się pod adresem *ftp://ftp.helion.pl/przyklady/unipig.zip*.

Skorowidz

I

J

K

L

M

N

O

P

R

S

Ś

T

U

W

Z

Unity
Przepisy na interfejs gry

Umiejętność projektowania i implementacji interfejsów użytkownika jest szczególnie ważna dla programistów gier. Aby gra mogła porwać i wciągnąć gracza, jej interfejs musi być zarówno funkcjonalny, jak i przyjemny dla oka. Idealnym narzędziem do tworzenia gier jest Unity: program, który pozwala na konstruowanie doskonałych, dynamicznych interfejsów. Szczególnie atrakcyjne jest łączenie ze sobą elementów graficznych i kodu tworzonego w języku C# — dzięki temu Unity pozwala na osiąganie imponujących wyników.

Niniejsza książka jest przeznaczona dla osób, które chcą wykorzystać silnik Unity do tworzenia gier wideo. Przedstawiono tu sposoby implementacji systemów interfejsu użytkownika i łączenia ich z pozostałymi składnikami gry. To zbiór przepisów na wykonanie określonych zadań, jednak najpierw omówiono tu zagadnienia podstawowe, a dopiero potem te nieco trudniejsze. Pozwala to na stopniowe doskonalenie umiejętności programistycznych. Podczas lektury książki warto zwrócić uwagę na odwołania do relacji między graczem a interfejsem gry — niezwykle ważny czynnik dla projektanta.

Lubisz grać? Spróbuj zaprojektować i napisać własną grę!

W książce omówiono:

- najważniejsze zagadnienia, o których trzeba pamiętać podczas tworzenia gier
- implementację liczników i pasków zdrowia, a także prezentację czasu w grze
- dodanie interaktywnych kontrolek, takich jak panele i suwaki
- tworzenie interaktywnych składników interfejsów i ich animowanie

Francesco Sapio jest inżynierem informatyki i automatyki, utalentowanym projektantem gier komputerowych i doświadczonym użytkownikiem programów graficznych. Zasłynął jako wybitny znawca oprogramowania Unity3D. Jest także muzykiem, ponadto przez kilka lat pracował jako aktor i tancerz. Uwielbia matematykę, filozofię, logikę i rozwiązywanie zagadek, ale jego największą pasją jest projektowanie i programowanie gier.

[PACKT] open source PUBLISHING community experience distilled

Helion

księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-2885-3

cena: 54,90 zł