Zbieranie punktów

9 Zbieranie punktów

9.1 Rozkładanie i obsługa kryształów

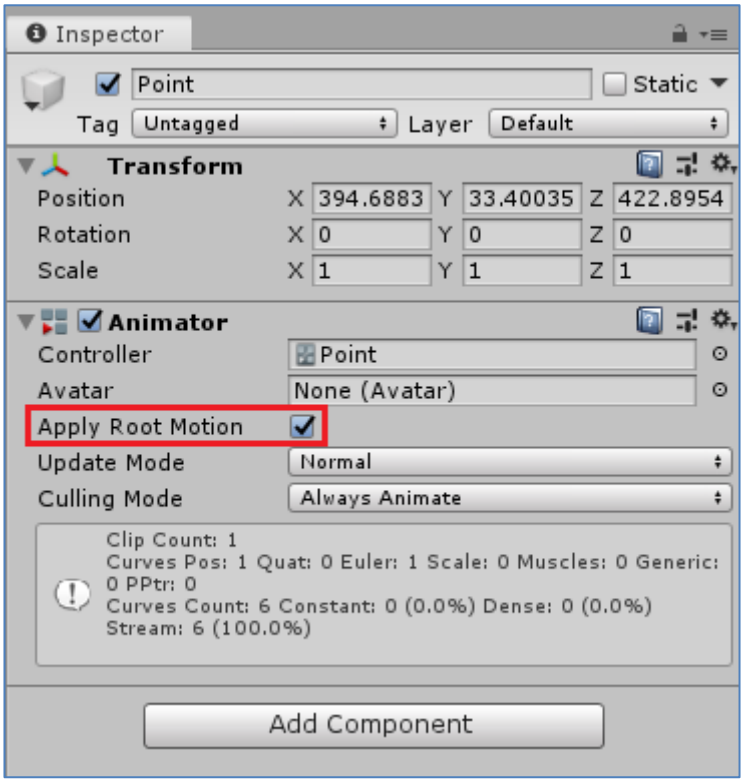
Gwiazdki, monety, kryształy, punkty… jak zwał tak zwał. Może i uda nam się znaleźć grę, w której niczego nie zbieramy. Jest to jednak warty uwagi aspekt, gdyż zebranie jak największej ilości takich obiektów jest satysfakcjonujące, przyczynia się do bardziej dogłębnego przetrząsania zakątków świata gry w celu odnalezienia tego ostatniego punktu oraz w przypadku niektórych gier wprowadza elementy rywalizacji, na zasadzie kto w krótszym czasie zbierze daną ilość monet.

Dobrze jest wiedzieć, jak w prosty sposób obsłużyć taki mechanizm w swojej grze. Nie jest to ani trudne, ani czasochłonne. Zaczynajmy! W rozdziale poprzednim stworzyliśmy przykładowy „kryształ" oraz nadaliśmy mu prostą animację. Jeśli nie czytasz jednym ciągiem, a skaczesz między rozdziałami, polecam wrócić o kilkanaście stron w celu doczytania tych informacji.

Tak więc mamy pojedynczy obiekt, posiadający konkretne właściwości. Jak pewnie wiesz, będziemy potrzebować takich obiektów trochę więcej, aby mogły być rozłożone na mapie. Zrobimy więc z naszego kryształu **prefab**.

Dla zapominalskich - czym jest prefab, tłumaczyłem w rozdziale drugim, gdzie też od razu utworzony został osobny katalog „**prefabs**", aby wszystko w naszym projekcie było na swoim miejscu. Utworzenie prefabrykatu nastąpi wraz z przeniesieniem obiektu (w moim przypadku) „**Point**" z panelu hierarchii do katalogu w panelu projektowym. Zanim jednak to uczynimy, kliknij, proszę w ten obiekt. Kilka rzeczy będzie wymagało drobnej modyfikacji.

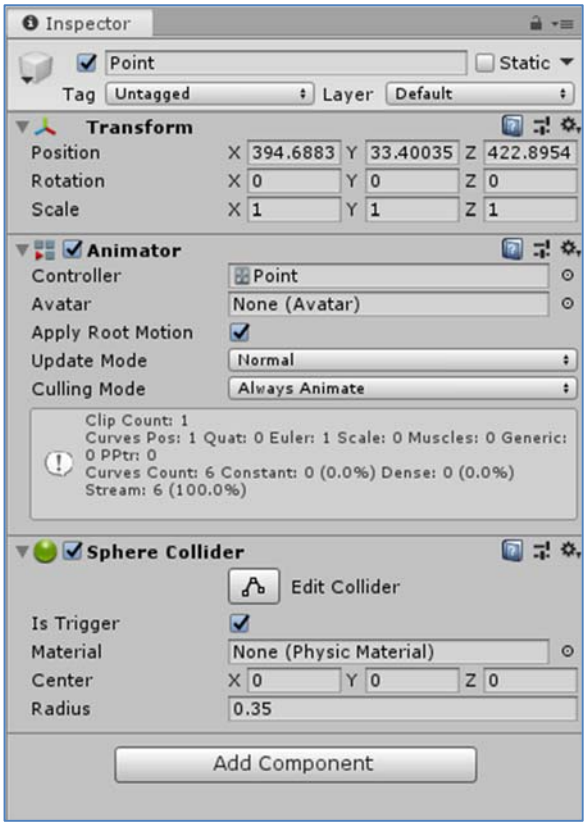
W komponencie „**Animator**", przydatne może się okazać zaznaczenie opcji „**Apply Root Motion**", o której nie powiedziałem Ci zbyt wiele w poprzednim rozdziale, gdyż chciałem, abyś dobrze zrozumiał tę kwestię na prostym przykładzie. Otóż stworzyliśmy kryształ, który ma swoją konkretną animację. Animacja ta działa na ściśle określonych wartościach położenia oraz rotacji.



Rysunek 9.1 Panel inspektora – obiekt „Point", komponent „Animator". Zaznaczenie opcji „Apply Root Motion"

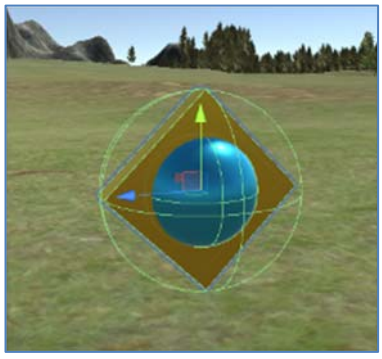
W momencie utworzenia bliźniaczego obiektu, właściwości animacji pozostaną niezmienione, a co za tym idzie, mimo rozstawienia kryształów po całym świecie, w chwili uruchomienia gry, wszystkie te obiekty znajdą się w jednym położeniu przypisanym w animacji, co da nam złudzenie, że wszystkie kryształy nagle zniknęły, a na swoim miejscu pozostał jedynie oryginał. By temu zapobiec, używamy wyżej wymienionej opcji, co spowoduje niejako wyliczenie wartości pozycji i rotacji względem nowego położenia kryształu.

Ostatnim krokiem będzie dodanie komponentu „**Sphere Collider**" (lub innego zderzacza, pasującego kształtem pod Twój obiekt), którego użyjemy jako **triggera**, by w momencie zetknięcia się z nim, kryształ został usunięty, a liczba zebranych punktów się zwiększyła.



Rysunek 9.2 Dodanie komponentu „Sphere Collider” pełniącego rolę triggera

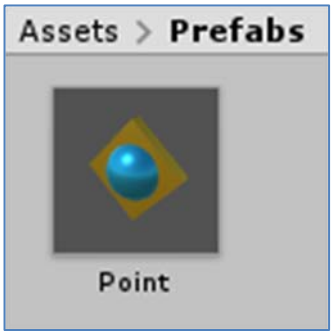
Dopasujmy wielkość collidera (triggera) tak, aby idealnie obejmował kształt obiektu.



Rysunek 9.3 Obiekt kryształu z dopasowaną siatką triggera

Jeżeli już mowa o colliderach, warto w tym miejscu zastanowić się, jakie są nasze preferencje, ponieważ kryształ, a właściwie jego obiekty składowe (sześcian i sfera) posiadają swoje domyślne collidery, przez co w momencie zebrania, odczuwalna będzie minimalna kolizja.

Jeżeli taki efekt jest dla Ciebie zbędny i wolałbyś tego uniknąć, wystarczy najzwyczajniej w świecie usunąć collidery podobiektów. Osobiście wolę, aby postać miała możliwość całkowitego przejścia przez kryształ, dlatego collidery zostały usunięte. Dobrze! Teraz możemy utworzyć już naszego prefaba:



Rysunek 9.4 Prefabrykat kryształu umieszczony w katalogu „Prefabs”

Pora na odrobinę zabawy. Przeciągnij na mapę kilka/kilkanaście sztuk i porozstawiaj je wedle własnego uznania:

Rysunek 9.5 Rozstawianie kryształów w świecie gry

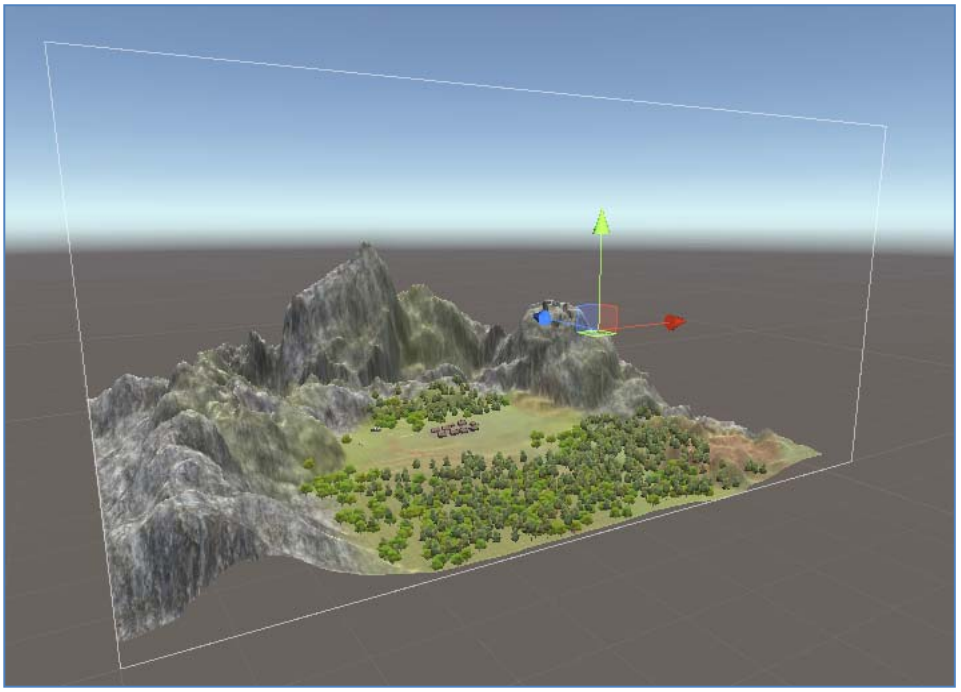
Po uruchomieniu gry, wszystkie kryształy powinny poprawnie wykonywać swoją zapętloną animację w położeniu, w którym zostały umiejscowione.

Zanim przejdziemy do skryptu, dokończmy wszystkie aspekty strony wizualnej, aby później nie skakać niepotrzebnie między środowiskiem a edytorem kodu. Mam tutaj na myśli coś zupełnie nowego. Mianowicie prosty **interfejs użytkownika**, którego zadaniem będzie informowanie gracza, jak wiele kryształów udało mu się zdobyć do tej pory.

Dokładniej w tę tematykę wgłębimy się w rozdziale **10.**, gdzie szczegółowo omówię proces tworzenia menu, jak i innych ciekawych urozmaiceń wyświetlanych na ekranie. Póki co przebrniemy przez tę kwestię dość powierzchownie.

9.1.1 Prosty interfejs wyświetlający punkty

Wybierz z górnego paska „**GameObject**” → ”**UI**” → ”**Canvas**”. Jakiś obiekt niby się utworzył, ale nie widać żadnych zmian… Kliknij dwukrotnie na nowo utworzony obiekt „**Canvas**” w panelu hierarchii.

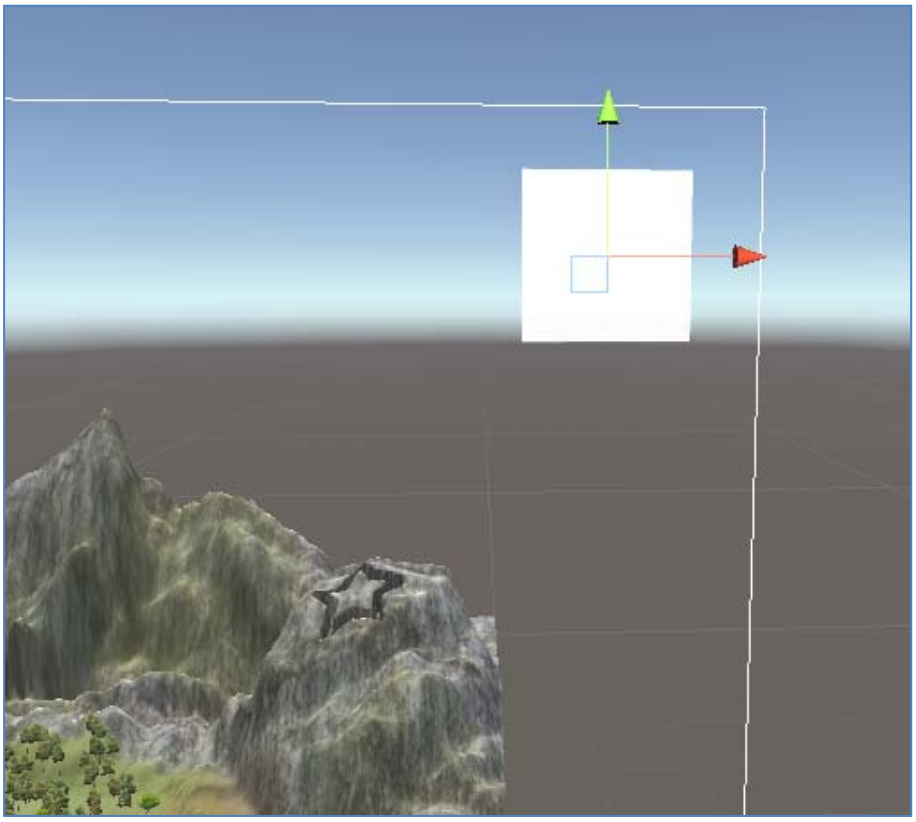


Rysunek 9.6 Nowo utworzony obiekt „Canvas”

Coś jednak jest! I to nie takie małe „coś”. Jest to w dosłownym tłumaczeniu „**płótno**”, które symbolizuje nasz ekran. Możemy dodawać na nim przeróżne

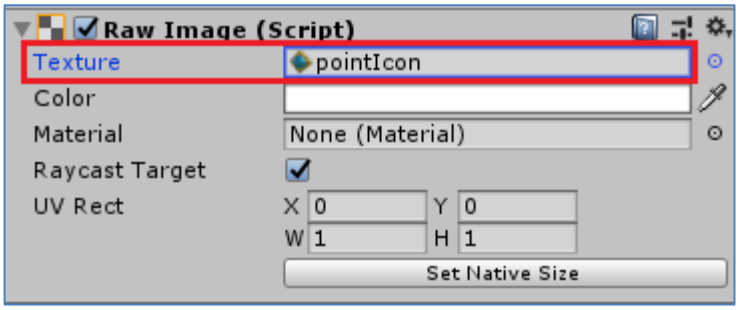
napisy, obrazki czy przyciski, a te będą wyświetlane dla nas podczas gry w taki sposób, jaki sobie życzymy. Ustawmy się teraz nieco bliżej płótna i wybierzmy z górnego paska „**GameObject**” → ”**UI**” → ”**Raw Image**”.

Jest to obiekt, do którego możemy wrzucić jakąś swoją grafikę, aby była np. wyświetlana jako obrazek na ekranie. Załóżmy, że chciałbym, aby interfejs zliczający punkty znajdował się w prawym górnym rogu ekranu. Przy pomocy strzałek przesuwam pusty obrazek w tamto miejsce. Coś w tym stylu.



Rysunek 9.7 Ustawienie obiektu „Raw Image” na płótnie

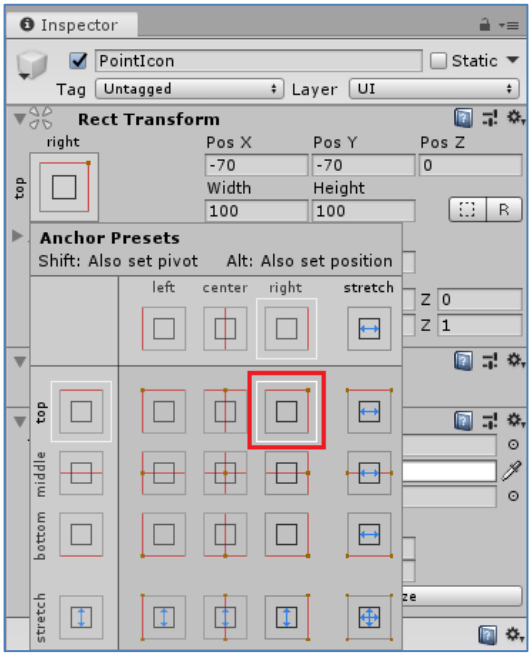
Teraz, aby wgrać do obiektu swoją grafikę, importujemy obraz do środowiska (najlepiej bezpośrednio do katalogu dla niego przeznaczonego, aby zachować porządek), a następnie w inspektorze obiektu „**Raw Image**” przeciągamy do rubryki „**Texture**” wybraną przez nas grafikę.



Rysunek 9.8 Panel inspektora – obiekt „PointIcon”, komponent „Raw Image”. Wybór własnej tekstury

Ja przygotowałem sobie na szybko coś na wzór kryształu z gry, aby gracz na pierwszy rzut oka mógł zorientować się, czego dany interfejs dotyczy. Korzystając z okazji, że jesteśmy już w tym inspektorze, ustawmy właściwość „**Rect Transform**” na prawy górny narożnik.

Służy to temu, aby poinformować dany obiekt, w jakim kierunku powinien „uciekać” w przypadku różnej rozdzielczości ekranów. Chcemy, aby znajdował się maksymalnie w prawym górnym rogu, stąd też ustawienia będą następujące.



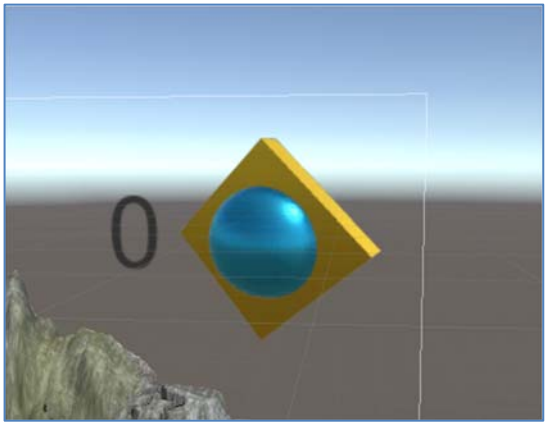
Rysunek 9.9 Modyfikacja domyślnego położenia w opcji „Rect Transform”

Jak widać, po uruchomieniu gry, nowy element interfejsu użytkownika na stałe zagościł na naszym ekranie:

Rysunek 9.10 Ikona kryształu umieszczona w prawym górnym rogu ekranu

Wiemy już więc, co zliczamy, ale nie mamy czym zliczać… Posłuży nam do tego najzwyklejsze pole tekstowe, którego wartość modyfikowana będzie przy pomocy skryptu. Wybierz z górnego paska „**GameObject**” → „**UI**” → „**Text**”.

Analogicznie do obrazka, ustaw nowo utworzone pole tekstowe w prawym górnym rogu z odrobiną odstępu od grafiki. Pamiętaj o nadaniu ustawień „**Rect Trasform**” na prawy górny róg (lub inny odpowiadający Twojemu umiejscowieniu). Wedle uznania możesz zwiększyć czcionkę, położenie tekstu w rubryce czy kolor. Jako wartość pola ustalimy po prostu „**0**”, co oznacza, że na razie nie mamy żadnego zebranego kryształu. Efekt:



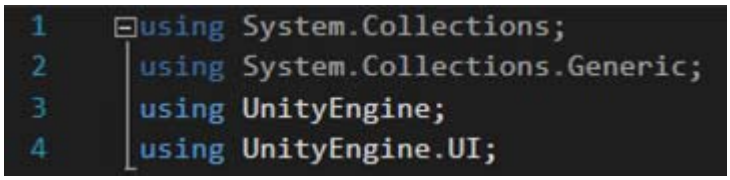
Rysunek 9.11 Uzupełnienie ikony kryształu o licznik zdobytych punktów

Ze strony wizualnej byłoby to wszystko. Teraz pora na uzupełnienie części logicznej całego mechanizmu.

9.1.2 Skrypt zliczający zebrane obiekty

W katalogu ze skryptami, utwórz nowy plik języka C#. Ja nazwę go np. „**PointCounter**”, aby w przyszłości wiedzieć, do czego służy. Otwórzmy skrypt w edytorze kodu. Co będzie nam potrzebne? Na pewno publiczna zmienna typu „**Text**”, do której podłączymy nasze pole tekstowe, aby możliwa była edycja jego wartości.

Uwaga! Typ „**Text**” zawarty jest w przestrzeni nazw „**UnityEngine.UI**”. Aby móc z niego skorzystać, konieczne jest dodanie owej przestrzeni w swoim skrypcie (**4**):



Rysunek 9.12 Dodanie przestrzeni nazw „UnityEngine.UI”

Poza tym, przydają się również dwie metody: „**OnTriggerEnter**” (**22**), która pozwoli wykryć czy dany kryształ został dotknięty przez gracza, czy też nie oraz druga typu „**int**” (liczby całkowite) o nazwie np. „**Points**” (**27**), która przechowa wszystkie kryształy (w tablicy) oraz zwróci wartość liczbową, jak dużo ich jeszcze pozostało.

Metody domyślne „**Start**” oraz „**Update**” na razie zostawmy. Będą nam potrzebne nieco później.

Metoda „**Points**” została podkreślona przez środowisko programistyczne z powodu braku instrukcji zwracającej wartość wewnątrz ciała („**return**”). Uzupełnijmy ją. Jako że każdy kryształ będzie miał przypisany ten skrypt do swojego inspektora, możemy obiekty te znaleźć właśnie po nim przy pomocy nowej metody „**FindObjectsOfType**” klasy „**Component**”.

```
7   public class PointCounter : MonoBehaviour
8   {
9       public Text counter;
10      public PointCounter[] objects;
11
12      void Start()
13      {
14
15      }
16
17      void Update()
18      {
19
20      }
21
22      public void OnTriggerEnter(Collider other)
23      {
24
25      }
26
27      public int Points()
28      {
29
30      }
31  }
```

Rysunek 9.13 Skrypt „PointCounter” - szkielet

Metoda ta zwraca tablicę, czyli zbiór obiektów tego konkretnego typu, jakie znajdują się na scenie. Tworząc tablicę obiektów typu „**PointCounter**” (**10**), jesteśmy w stanie przechwycić zwróconą przez metodę tablicę (**68**). Na koniec zwracamy rozmiar tej tablicy (**69**), aby mógł zostać użyty w innej funkcji.

```
66      public int Points()
67      {
68          objects = Component.FindObjectsOfType<PointCounter>();
69          return objects.Length;
70      }
```

Rysunek 9.14 Ciało metody „Points”

Troszkę zagmatwane, wiem. Staram się opisywać to wszystko jak najprościej. Jeżeli masz problem, spróbuj przeanalizować ten kod, a później ponownie przeczytać opis. Dodatkowo radzę uważać na poprawny zapis, ponieważ metod o podobnej nazwie istnieje dużo więcej, a przypadkowe zastosowanie metody „**FindObjectOfType**" zamiast „**FindObjectsOfType**" spowoduje błąd, który może być trudny do zauważenia, co może kosztować nas sporo czasu.

Tymczasem przejdźmy do zdefiniowania ciała metody „**OnTriggerEnter**". Jak w przypadku drzwi, najpierw sprawdzimy, czy obiektem, który wywołał kolizję, jest gracz (**39**). Jeżeli tak, przejdź do wykonywania dalszych poleceń. W przeciwnym wypadku, opuść całą funkcję (sam „**return;**" bez niczego innego).

```
37      public void OnTriggerEnter(Collider other)
38      {
39          if(other.tag == "Player")
40          {
41
42          }
43          else
44          {
45              return;
46          }
47      }
```

Rysunek 9.15 Szkielet ciała metody „OnTriggerEnter"

Załóżmy, że to gracz wszedł w interakcję z kryształem. Należałoby sprawdzić kolejne ewentualności. Warunek – jeżeli liczba elementów typu „**Point**" (zwrócona z funkcji „**Points**") jest równa **1**, to wykonaj… coś (to za chwilę). W przeciwnym wypadku **usuń** ten obiekt i zwiększ wartość na liczniku.

Do usuwania obiektów w czasie rozgrywki służy funkcja „**Destroy**", której parametrami są: **gameObject**, czyli ten konkretny obiekt, na którym pracuje skrypt oraz czas, po jakim ma nastąpić usunięcie (w sekundach). Jeśli chodzi o zapis wartości w liczniku na ekranie, skorzystamy tutaj z metody konwertującej („**Convert**"), która to zamieni tekst w wartość liczbową. Ta następnie (już jako liczba) zostanie zwiększona o jeden przy pomocy operatora „++" i ponownie przypisana polu tekstowemu metodą „**ToString**".

Dlaczego **Points() == 1**, a nie **Points() == 0** - zapytasz… Dlatego, że przy zetknięciu się z ostatnim możliwym obiektem, wielkość tablicy kryształów będzie miała właśnie wartość **1** (ten ostatni obiekt to ten, który właśnie próbujemy zebrać).

```
37    public void OnTriggerEnter(Collider other)
38    {
39        if(other.tag == "Player")
40        {
41            if (Points() == 1)
42            {
43
44            }
45            else
46            {
47                Destroy(gameObject, 0.1f);
48
49                int currentValue = Convert.ToInt32(counter.text);
50                currentValue++;
51
52                counter.text = Convert.ToString(currentValue);
53            }
54        }
55        else
56        {
57            return;
58        }
59    }
```

Rysunek 9.16 Warunek sprawdzający ilość punktów

Teraz zastanówmy się nad tym, co konkretnie miałoby się wydarzyć po zebraniu wszystkich kryształów. Jeżeli gra podzielona jest na poziomy, to mogłoby nastąpić załadowanie kolejnego i kolejnego, aż do ostatniego, który przekieruje do napisów końcowych czy głównego menu.

Póki co jednak pójdziemy na łatwiznę i ustalimy, że wraz z zebraniem ostatniego punktu na mapie, kryształ zostanie usunięty, licznik ukryty, a wyświetlona zostanie informacja w konsoli o treści „Gratulacje! Udało Ci się zebrać wszystkie kryształy!”.

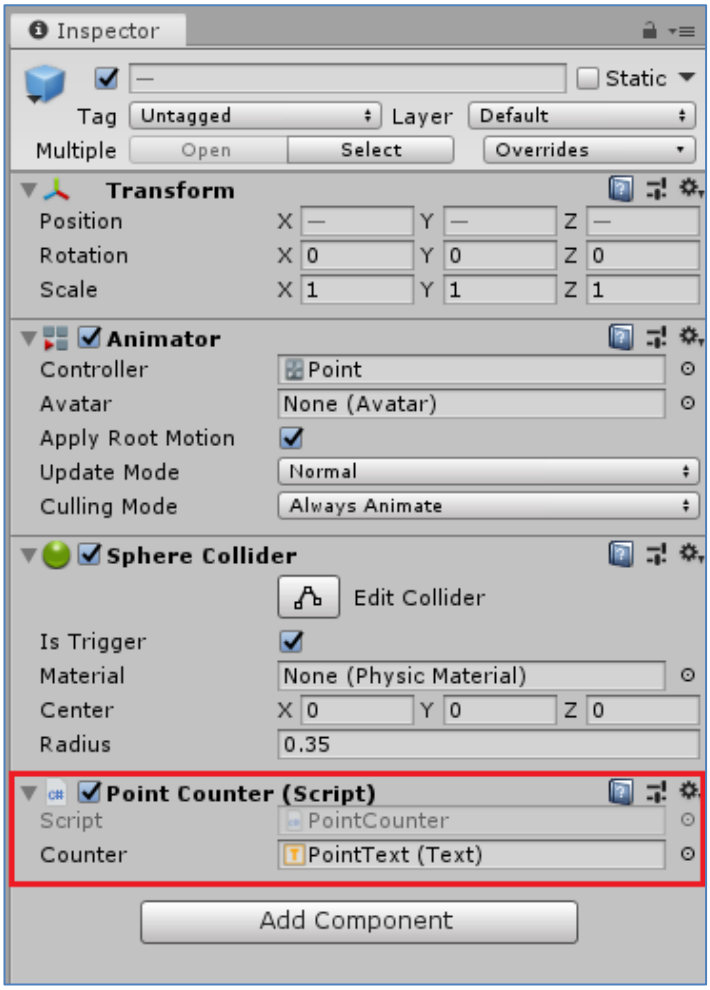
```
41          if (Points() == 1)
42          {
43              Destroy(gameObject, 0.1f);
44
45              counter.enabled = false;
46
47              Debug.Log("Gratulacje! " +
48                  "Udało Ci się zebrać " +
49                  "wszystkie kryształy! ");
50          }
```

Rysunek 9.17 Ciało warunku sprawdzającego ilość punktów dla wartości: Points() == 1

Tak by się to prezentowało. Jak można zauważyć, z każdym kolejnym rozdziałem, skrypty stają się coraz bardziej rozbudowane, a co za tym idzie, są trudniejsze w odbiorze dla początkujących twórców.

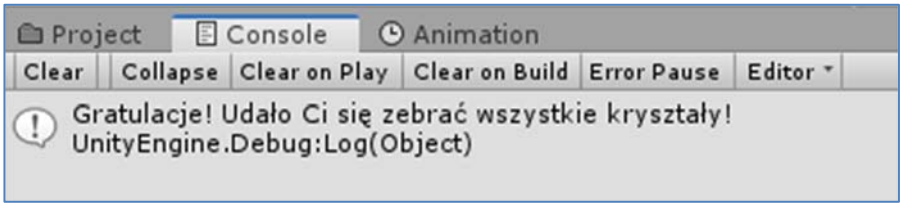
Bardzo Cię jednak proszę, abyś się nie zrażał, gdyż wszystko jest dla ludzi i po dłuższym obyciu się z pisaniem skryptów, jak i samym budowaniem świata gry, wszystko staje się jasne.

Pora podpiąć skrypt pod obiekty. Celowo nie zrobiłem tego wcześniej przed stworzeniem prefaba, ponieważ chciałem podzielić się z Tobą fajną ciekawostką. Aby dodać komponenty do inspektora wielu obiektów naraz, wystarczy zaznaczyć wszystkie te obiekty w panelu hierarchii (z pomocą klawisza **Shift** lub **Ctrl**), a następnie przeciągnąć komponent (w naszym przypadku skrypt) do inspektora. Teraz, należy każdemu z tych kryształów dodać obiekt typu „**Text**”, który pełni rolę naszego licznika. Odbywa się to na tej samej zasadzie. Zaznaczamy wszystkie kryształy i przeciągamy!



Rysunek 9.18 Podłączenie skryptu wraz z komponentem „Text” do wszystich kryształów na scenie

Przetestujmy uzyskane efekty. Zbiorę teraz te kilka kryształów i zobaczymy…



Rysunek 9.19 Ustalony komunikat konsolowy

Jest! Świetnie. Na tym w zasadzie można by było zakończyć ten rozdział, ale pomyślałem, że warto będzie nieco rozbudować nasz skrypt o dodatkowe funkcjonalności.

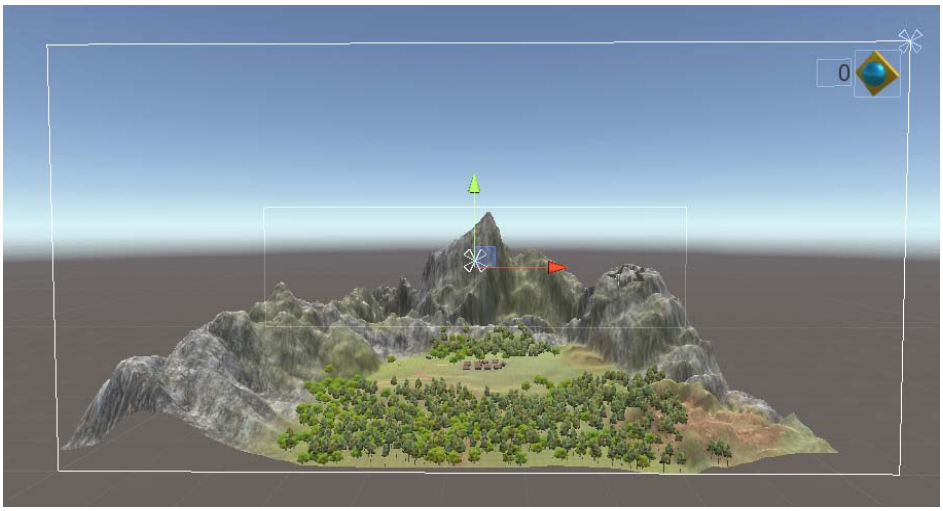
Oczywiście nie jest to konieczne, ale dość ciekawe. Do rzeczy! Mam na myśli stworzenie prostego systemu odmierzania czasu od rozpoczęcia rozgrywki do zebrania ostatniego punktu. Spróbujmy zrobić coś takiego.

9.1.3 Odmierzanie czasu zbierania kryształów

Ponownie, jak w przypadku poprzedniego podrozdziału, rozpoczniemy od strony wizualnej. Pójdziemy o krok dalej i zamiast w konsoli, spróbujemy wyświetlić końcowy komunikat na ekranie. Potrzebne będzie jeszcze jedno pole tekstowe na płótnie.

Przez całą rozgrywkę będzie ono nieaktywne, a ukaże się jedynie pod koniec wraz z zebraniem ostatniego punktu. Wyświetlimy na nim gratulacje, jak w poprzednim przypadku oraz czas, jaki był potrzebny na zebranie wszystkich obiektów.

Warto oczywiście odrobinę powiększyć rozmiar pola tekstowego oraz czcionkę, by podczas rozgrywki wszystko było widoczne. Nie będzie to i tak ekran wygranej wysokich lotów, ale chodzi mi o sam mechanizm.



Rysunek 9.20 Dodanie nowego pola tekstowego na płótnie

Kwestie graficzne i estetyczne pozostawiam już Tobie. Pole zostawiamy puste, ponieważ tekst wpiszemy przy pomocy skryptu. Tak więc wróćmy do niego… Potrzebne będzie jeszcze jedno publiczne pole typu „**Text**” (**10**), do którego podepniemy nowo utworzone pole tekstowe, a także prywatne zmienne typu liczbowego (**12-14**), którymi obsłużymy pomiar czasu oraz odpowiednio sformatujemy dane wyjściowe.

```
 9      public Text counter;
10      public Text time;
11
12      private float timer = 0;
13      private int minutes = 0;
14      private int seconds = 0;
15
16      public PointCounter[] objects;
```

Rysunek 9.21 Dodanie nowych pól obsługujących odmierzanie czasu

Jako iż są to wartości liczbowe, na starcie przyrównałem je do zera, aby uniknąć późniejszych problemów z ich modyfikacją.

Zmienna „**timer**” obsługiwana będzie przez metodę „**deltaTime**” klasy „**Time**” w domyślnej metodzie „**Update**”, co pozwoli nam uzyskać rzeczywisty czas, jaki będzie potrzebny na zebranie kryształów (**27**). Zmienna typu całkowitego „**seconds**”, będzie zbierała z licznika części całkowite oznaczające sekundy (**28**). Poniżej zastosujemy warunek (**30**), który w momencie osiągnięcia przez sekundy wartości **60**, doda jedną minutę (**32**) i wyzeruje licznik (**33**). Dość logiczne.

```
25      void Update()
26      {
27          timer += Time.deltaTime;
28          seconds = (int)timer;
29
30          if (seconds == 60)
31          {
32              minutes++;
33              timer = 0;
34          }
35      }
```

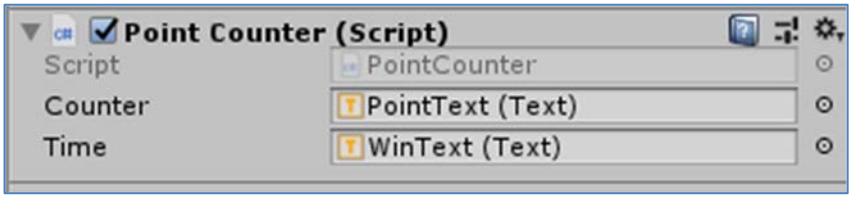
Rysunek 9.22 Ciało metody domyślnej „Update”

Na koniec pozostało zmodyfikować ciało warunku sprawdzającego, czy liczba pozostałych obiektów to **1**. Nastąpi podmiana zawartości pola tekstowego „**time**” (**47**). Początek zostawimy taki sam jak w przypadku konsoli, lecz dodamy jeszcze czas, który upłynął wraz z jednostką (**49**).

```
41          if (Points() == 1)
42          {
43              Destroy(gameObject, 0.1f);
44
45              counter.enabled = false;
46
47              time.text = "Gratulacje! " +
48                  "Udało Ci się zebrać wszystkie kryształy! " +
49                  "Czas: " + minutes + " minut " + seconds + " sekund!";
50          }
```

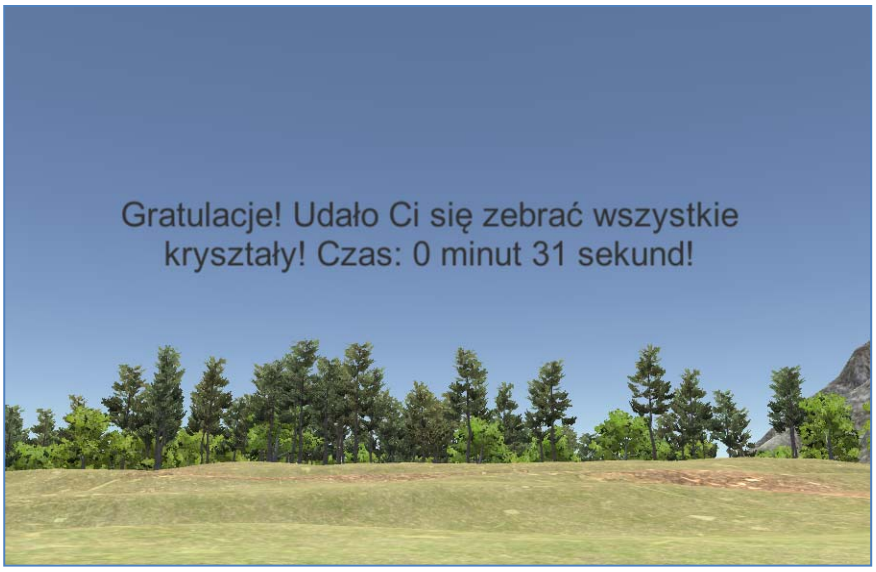
Rysunek 9.23 Dostosowanie wyświetlania komunikatu na płótnie

Tym oto sposobem zbliżamy się do końca. Pamiętaj jednak, że dodaliśmy nowe pole publiczne, które należy uzupełnić o dodane przed chwilą pole tekstowe. Jak poprzednio, zaznacz wszystkie kryształy i jednorazowo przenieś odpowiedni obiekt z panelu hierarchii do inspektora:



Rysunek 9.24 Podpięcie nowego pola tekstowego w komponencie „Point Counter” dla wszystkich kryształów na scenie

Pora na testy! Nie mogę się doczekać ☺



Rysunek 9.25 Komunikat wyświetlany na płótnie

To działa! Wynik nawet niezły. Ciekaw jestem, jaki udałoby się uzyskać Tobie. Podsumowując ten rozdział, na pewno warto zauważyć, że pojawiło się w nim mnóstwo nowych informacji i ciekawostek, które odrobinę podwyższyły poziom Twojej wiedzy w zakresie tworzenia gier.

Brawo! Skoro już było nam dane pobawić się płótnem i jego komponentami… Co powiesz, aby zrobić w końcu jakieś menu dla naszej gry? Ja myślę, że jest to nawet konieczne! Zapraszam Cię na kolejną stronę.