

Lekcja 1: PLATFORMY I AWATAR**Cel i zakres pracy**

Celem ćwiczenia jest utworzenie podstawowej sceny 2D wraz z platformami, animowanym obiektem gracza, a także poruszanie awatarem przy pomocy klawiatury oraz myszki. W przypadku realizacji zdalnej, samodzielnym zadaniem sprawdzającym stopień przyswojenia materiału będzie rozbudowa poziomu o kolejne platformy i nagranie filmiku pokazującego przejście od początku do końca poziomu.

Uwaga

Projekt ten będzie rozwijany na kolejnych zajęciach. Po skończeniu zajęć USUŃ folder Library, SKOMPRESUJ projekt (zip) i SKOPIUJ go na dowolny nośnik lub dysk sieciowy.

Utworzenie projektu

1. Jeżeli jeszcze nie masz konta *Unity*, to załóż je na stronie <https://unity.com/>.
2. Jeżeli na komputerze nie ma zainstalowanego *Unity*, to zainstaluj najnowszą wersję o długim czasie wsparcia LTS (ang. *long term suport*) **2022.3.10f1**.
3. Uruchom *Unity Hub* i w zakładce *Projects* utwórz nowy projekt (*New project*). Przy kolejnym uruchomieniu *Unity Hub* będzie można wybrać projekt spośród listy dostępnych projektów.

Uwaga

Realizacja projektu w innej wersji *Unity* może utrudnić realizację zadań (potencjalne różnice w funkcjonalności) oraz spowodować obniżenie oceny!

4. Na samej górze okna dialogowego (rys. 1) wybierz wersję **2022.3.10f1** oraz rodzaj projektu **2D Cross-Platform (URP)** i podaj nazwę **2D Platformer I Nazwisko**, gdzie I jest inicjałem imienia (należy podać nazwisko/nazwiska **bez polskich liter**), wskaż własny folder na dysku **D:** (**nie pulpit Windows**) i utwórz projekt (może potrwać dłuższą chwilę).

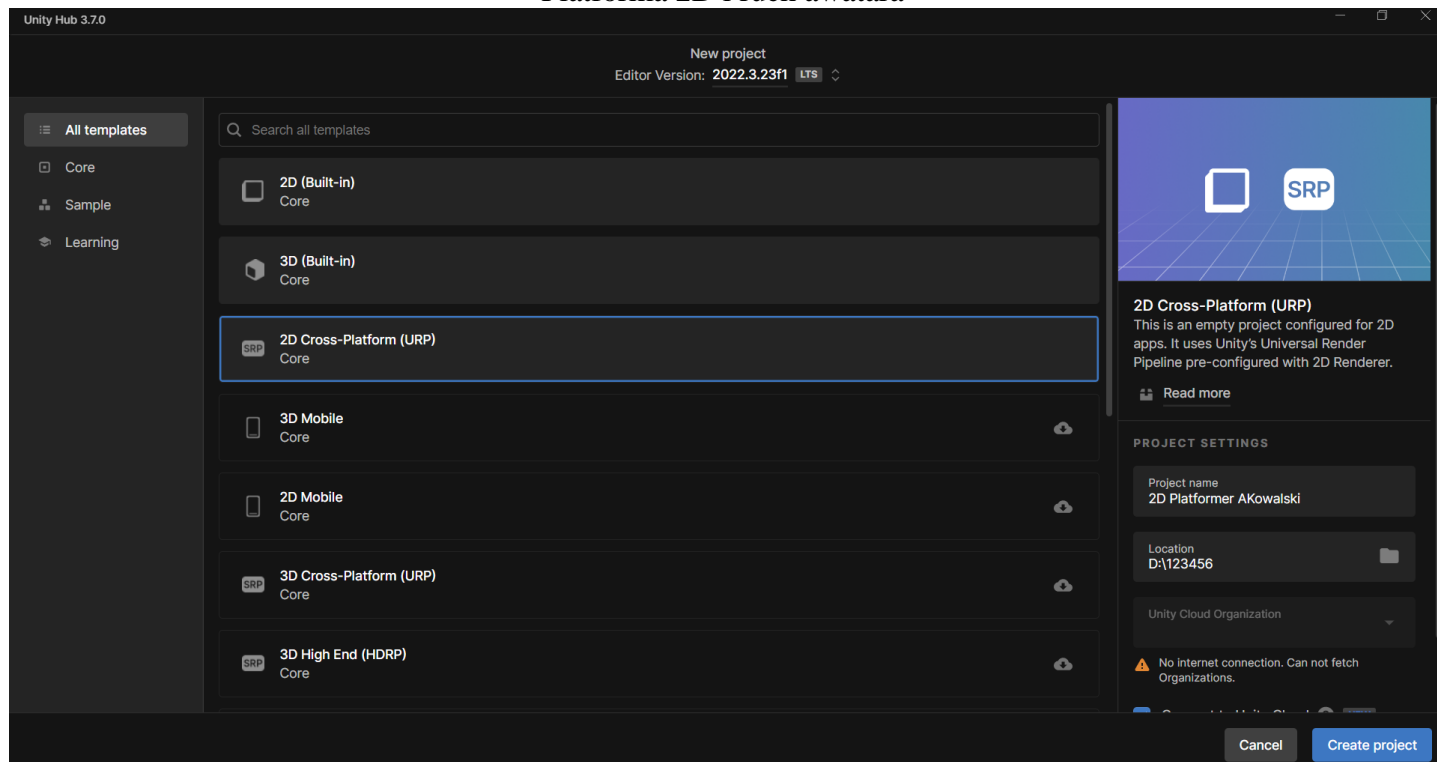
Uwaga

Projekty typu 2D URP (*universal rendering pipeline*) pozwalają na uzyskanie o wiele ciekawszych efektów oświetlenia w porównaniu z wbudowanymi (ang. *built-in*) projektami typu 2D, które wykorzystują wbudowany potok renderowania. Mimo, iż odbywa się to kosztem braku kompatybilności z niektórymi starszymi zasobami, to warto używać trybu 2D URP.

5. Edytor *Unity* oferuje kilka predefiniowanych układów okien (*Window*→*Layouts*). Instrukcja będzie opierała się na układzie domyślnym (*Default*).
6. W oknie *Project* wskaż folder *Assets*→*Scenes* i zmień nazwę sceny na **Level 1**.
7. Zapisz projekt (*File*→*Save Project*) oraz scenę (*File*→*Save* ≡ *Ctrl+S*).

Uwaga

W czasie pracy należy systematycznie zapisywać **zarówno scenę jak i cały projekt**.



Rys. 1. Okno tworzenia nowego projektu w Unity.

Przygotowanie zasobów

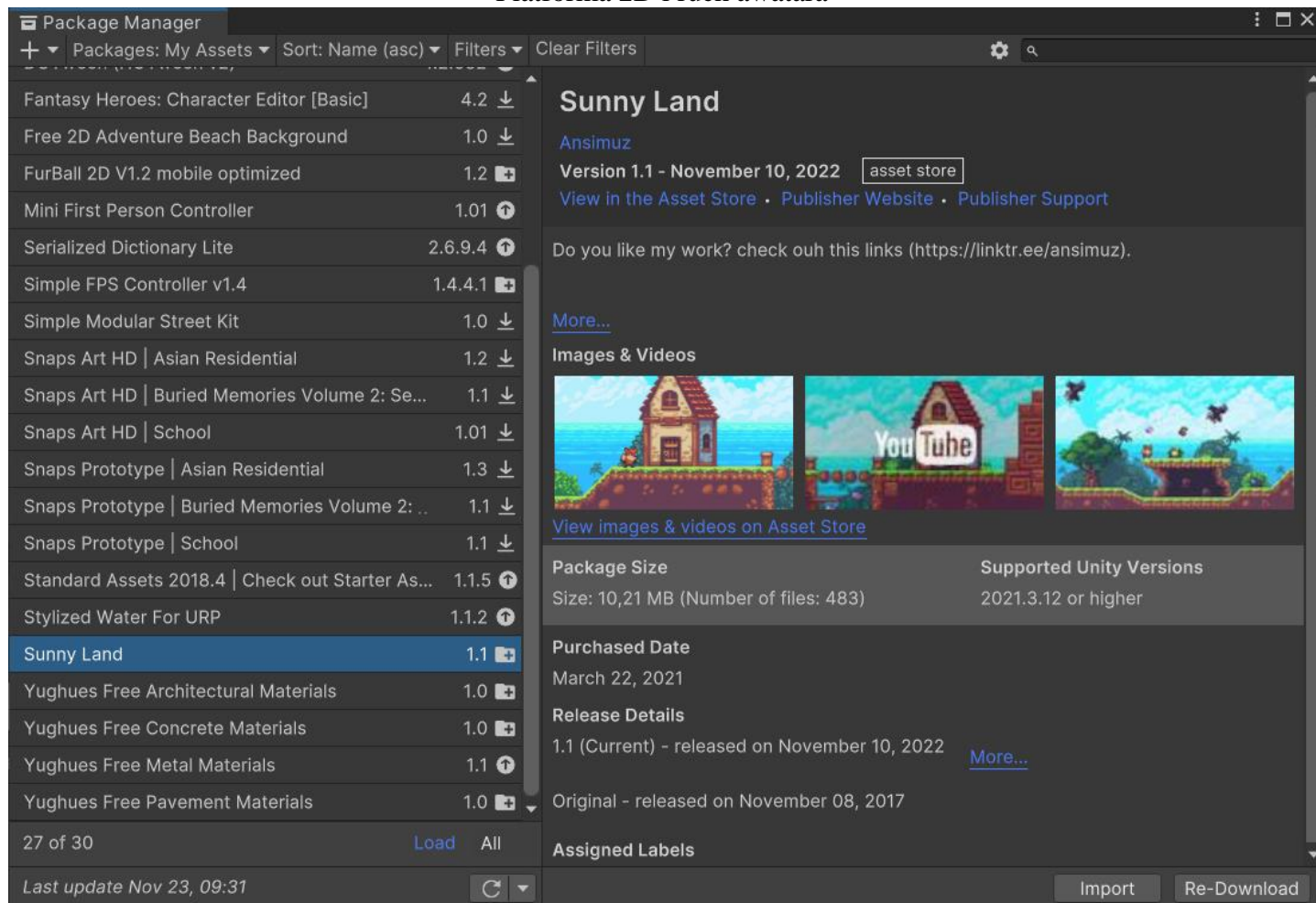
Uwaga

Instrukcja została oparta na darmowym pakiecie *Sunny Land v1.2* (widocznym jednak jako v.1.1). Po zakończeniu ćwiczenia będzie można łatwo zmienić pakiet zasobów i/lub dodać nowe. W niektórych miejscach instrukcji mogą pojawić się informacje odnośnie innych pakietów w celu ukazania ewentualnych różnic.

8. Jeżeli nie masz jeszcze zakupionego pakietu *Sunny Land v1.2*, otwórz sklep *AssetStore* w przeglądarce (*Window* → *AssetStore* lub bezpośrednio w przeglądarce pod adresem <https://assetstore.unity.com/>) i zaloguj się do swojego konta *Unity*. Wyszukaj pakiet *Sunny Land v1.2* i wybierz opcję *Add to My Assets*, a następnie *Open in Unity*.
9. Otwórz zarządcę pakietów (*Window* → *Package Manager*), z rozwijanej listy na górnej belce wybierz opcję *My Assets* (zamiast *In Project*) i wskaż pakiet *Sunny Land*. W przypadku posiadania starszej wersji pakietu zaktualizuj ją! Wybierz opcję *Download* lub *Import* w prawym dolnym rogu okna (rys. 2).
10. Upewnij się, że zaznaczone są wszystkie (interesujące Cię elementy zasobu) w wyświetlonym oknie importu. W przypadku pakietu *Sunny Land* nie należy importować folderu *Scenes* (rys. 3). Po wybraniu przycisku *Import*, zaimportowany zasób pojawi się w oknie *Projektu* (rys. 4).

Uwaga

W przypadku niektórych starszych pakietów mogą się pojawić błędy związane z niekompatybilnością ich niektórych składowych, których nie należy w takim przypadku importować. Przykładowo, w pakiecie *FurBall 2D* należy odznaczyć opcje *FurBall2D_mobile* oraz *Standard Assets*, gdyż powodują one błędy kompilacji w nowszych wersjach *Unity*.



Rys. 2. Przykładowe okno zarządcy pakietów. W prawym dolnym rogu widoczny jest przycisk importu.

11. Po zaimportowaniu zasobu poczekaj na ewentualną kompilację załączonych skryptów.

Uwaga

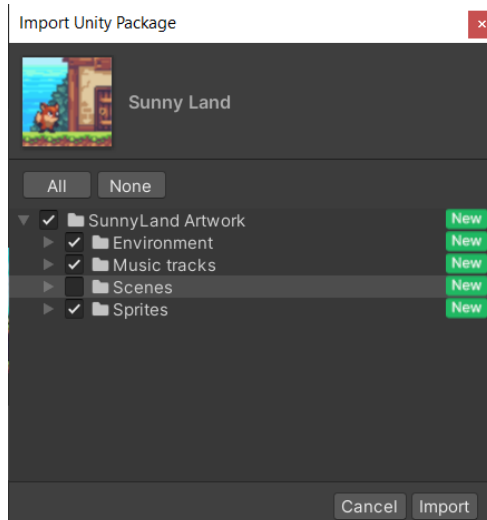
Jeżeli pojawią się błędy możesz spróbować je naprawić lub usunąć zasób i wybrać inny, bez błędów. Przykładowo, po imporcie **pełnego** pakietu *FurBall 2D* pojawiają się błędy związane z odwołaniami do niewspieranych już systemów BlackBerry i Windows Mobile w skrypcie *CrossPlatformInput.cs* (które można oczywiście ręcznie usunąć).

12. Po zaimportowaniu wszystkich potrzebnych zasobów zamknij okno sklepu i zapisz bieżącą scenę (*Ctrl+S*) oraz projekt (*File→Save Project*).

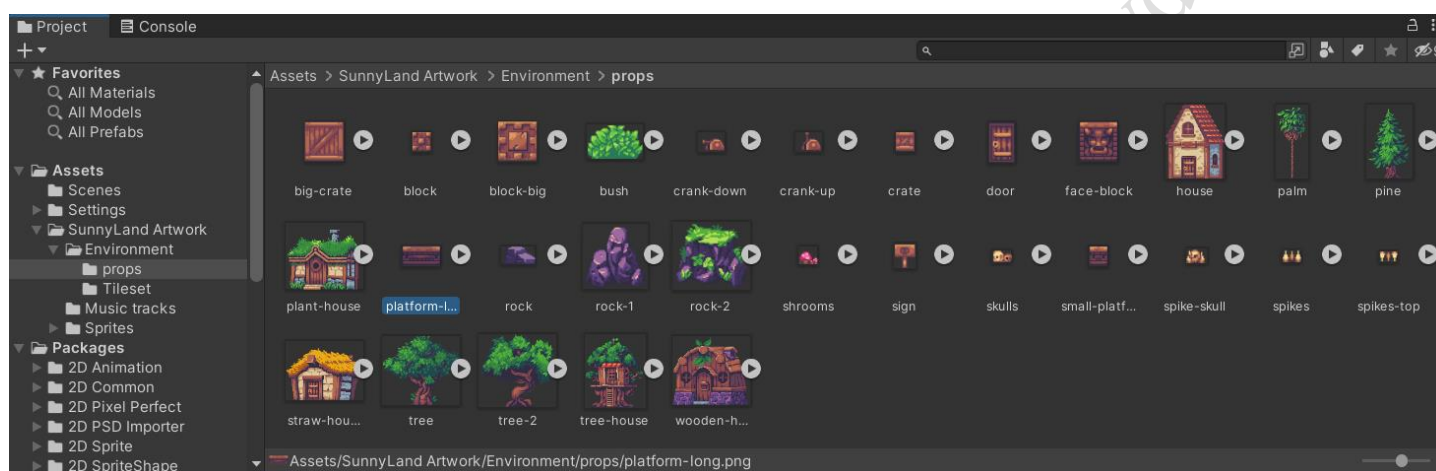
Uwaga

Nabierz nawyku regularnego zapisywania zarówno projektu jak i samej sceny co kilka/kilkanaście minut pracy!

Platforma 2D i ruch awatara



Rys. 3. Importowanie pakietu *Sunny Land*.



Rys. 4. Zaimportowane zasoby w oknie projektu.

I. Utworzenie obiektu platformy i gracza (30%)

13. Utworzenie platformy na scenie


- W oknie *Project* odszukaj w zaimportowanych zasobach folder zawierający użyteczne grafiki (*Assets*→*SunnyLand Artwork*→*Environment*→*Props* (rys. 4), zaznacz obiekt *platform-long* i w oknie *Inspector* ustaw właściwą skalę obiektu *Sprite Mode*→*Pixels Per Unit*=16.

Uwaga



Pamiętaj, aby w całym projekcie należy konsekwentnie ustawić właściwą skalę, dostosowaną do wielkości grafiki rastrowej, dla wszystkich używanych obiektów. W przypadku pakietu *Sunny Land* WSZYSTKIE UŻYWANE zasoby graficzne powinny mieć ustawioną skalę na 16.

- Wskaż obiekt platformy (*platform-long*) i przeciągnij go na środek sceny. W oknie *Inspector* wyzeruj położenie obiektu na scenie (*Transform*→*Position XYZ*=0 lub *Reset* z menu komponentu *Transform*).

Uwaga

W przypadku obiektów o charakterze rastrowym (np. *Sunny Land*) staraj się nie zmieniać proporcji obiektów, blokując ich skalę (*Transform*→*Scale*) ikonką .

Platforma 2D i ruch awatara




- c. W oknie *Hierarchy* na lewo od wszystkich obiektów znajduje się ikona oka , która pozwala na wyłączenie wyświetlania na scenie poszczególnych obiektów lub reprezentujących je ikon. Wyłącz wyświetlanie ikon *Main Camera* i *Global Light 2D*, tak aby nie przeszkadzały w edycji sceny.
- d. W oknie *Hierarchy* kliknij na główną kamerę (*Main Camera*) i w oknie *Inspector* dostosuj rozmiar pola widzenia *Camera*→*Size* tak, aby obiekt był dobrze widoczny na scenie (rys. 5).
- e. W oknie *Project* utwórz nowy folder *Assets*→*Prefabs* (→*Folder*), który będzie zawierał wszystkie utworzone szablony.

Uwaga

Odpowiednia struktura folderów pozwala na utrzymania porządku w projekcie.

- f. Przeciągnij nazwę obiektu (*platform-long*) z okna *Hierarchy* do nowo utworzonego folderu *Assets*→*Prefabs*, tworząc w ten sposób szablon. **Zauważ zmianę koloru nazwy platformy w oknie *Hierarchy*, która wskazuje, że platforma na scenie jest obecnie instancją szablonu.**



Uwaga

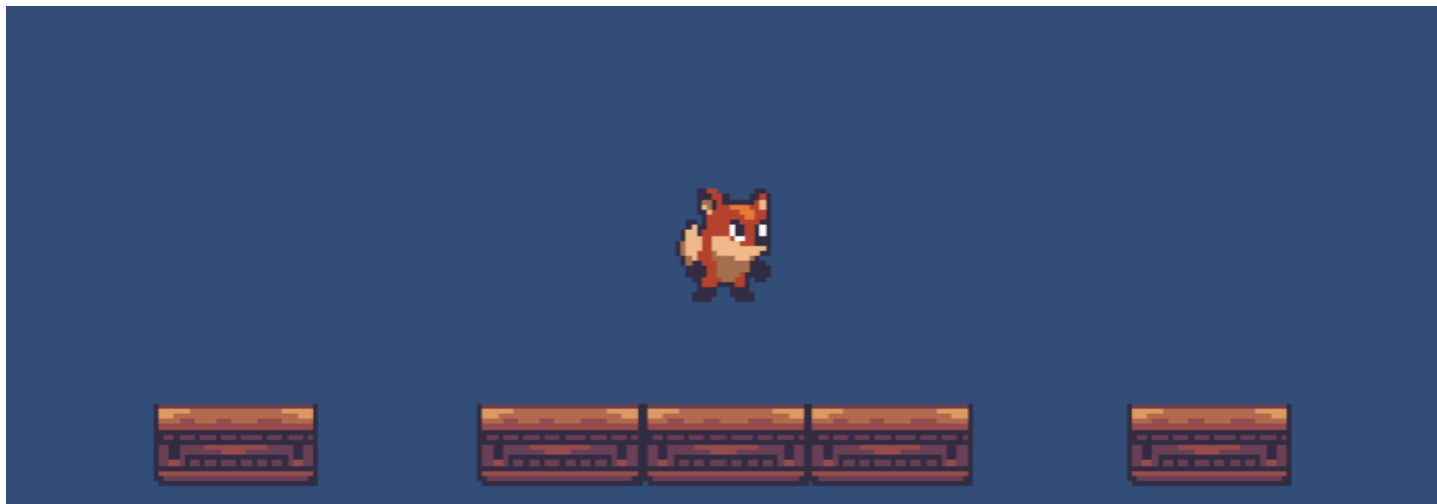
Szablony (ang. *prefabs*) odgrywają niezwykle istotną rolę w edytowaniu sceny, pozwalają na wielokrotne wykorzystanie zasobów i jednoczesną zmianę właściwości wszystkich swoich instancji na scenie! Jeżeli obiekt nie jest instancją szablonu, to w oknie *Hierarchy* jego nazwa ma szary kolor i jest oznaczony krawędziowym modelem sześcianu . Z kolei nazwy instancji szablونów są oznaczone kolorem niebieskim oraz ikoną sześcianu z pełnymi bokami:  (*prefab*) lub  (*variant prefab*).

- g. Dodaj do sceny kilka dodatkowych instancji szablonu platformy (przeciągając je na scenę z folderu *Assets*→*Prefabs*) położonych na tej samej wysokości co pierwsza platforma (rys. 5).

Uwaga

Tę samą wysokość platform możesz ustawić zaznaczając je wszystkie platformy i ustawiając ich współrzędną *Transform*→*Position Y=0*. W przypadku krótkich platform o charakterze rastrowym (np. *Sunny Land*) lepiej jest dodać kilka kolejnych platform niż wydłużać istniejącą (rys. 5).


- h. Wybierz okno *Game*, wybierz z rozwijanej listy opcję *Maximized* (zamiast *Normally*).
- i. Uruchom grę (*Edit*→*Play*≡*Ctrl+P*, ) i zaobserwuj dodane platformy (rys. 5, rys. 6a).
- j. Wyłącz grę (*Edit*→*Play*≡*Ctrl+P*, ). Od tej pory możesz regularnie sprawdzać wygląd swojej gry!
- k. W razie potrzeby dostosuj pole widzenia kamery, tak aby po uruchomieniu gry widoczne były wszystkie platformy (rys. 5).


Rys. 5. Przykładowa scena z platformą z pakietu *Sunny Land*.

14. Utworzenie obiektu gracza

- a. Znajdź w zasobach dowolną grafikę bohatera gry (awatara gracza). W przypadku pakietu *Sunny Land* będzie to np. *Assets*→*SunnyLand Artwork*→*Sprites*→*player*→*idle*→*player-idle-1*.
- b. Przeciągnij grafikę na scenę powyżej środkowej platformy (rys. 5). Zauważ, że w okno *Hierarchy* został utworzony obiekt awatara. Komponent *Sprite Renderer* obiektu (widoczny w oknie *Inspector*) odpowiedzialny jest za jego wyświetlanie na scenie.


Uwaga

Alternatywną metodą jest utworzenie pustego obiektu (*GameObject*→*Create Empty*≡*Ctrl+Shift+N*), dodanie do niego komponentu *Sprite Renderer* (*Component*→*Rendering*→*SpriteRenderer*) i ustawienie w oknie *Inspector* grafiki awatara jako parametru *Sprite Renderer*→*Sprite*→.

- c. W oknie *Hierarchy* zmień nazwę obiektu gracza na **Player**.
- d. Uruchom grę – obiekt gracza pozostaje nieruchomy względem platformy.
- e. Dodaj do obiektu gracza komponent fizyki ciała sztywnego (*Component*→*Physics 2D*→*Rigidbody 2D* lub w oknie *Inspector* →*Physics 2D*→*Rigidbody 2D*).

Uwaga

Komponent ten jest odpowiedzialny za symulację fizyki obiektów, w tym m.in. oddziaływaniu siły ciężkości i kolizje.

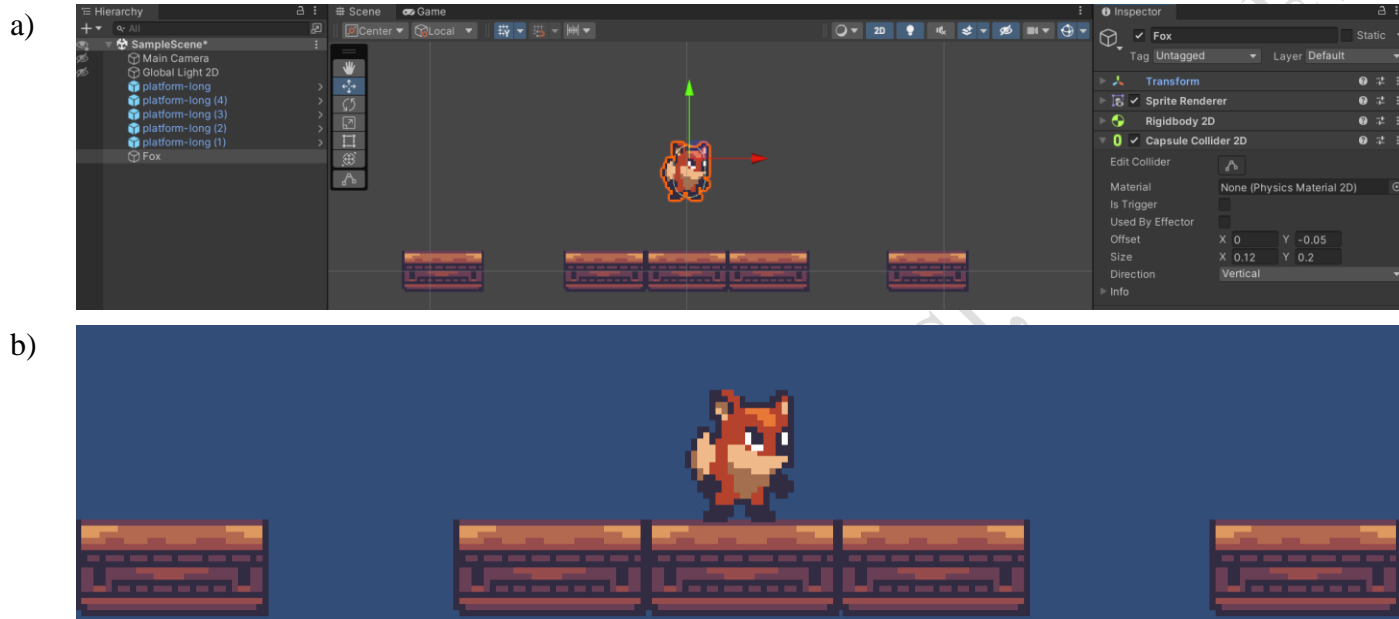
- f. W dodanym komponencie zaznacz właściwość *Rigidbody 2D*→*Constraints*→*Freeze Rotation Z*, aby zapobiec niepożądanemu obracaniu się obiektu.
- g. Uruchom grę i zaobserwuj spadanie awatara pod wpływem działającej na niego siły ciężkości. Platforma nie stanowi tu przeszkody, gdyż obiekt gracza nie ma jeszcze komponentu kolizji.
- h. Dodaj do obiektu gracza komponent kolizji (*collider*), który powinien być zbliżony kształtem do postaci gracza, np. prostokąta (*Component*→*Physics 2D*→*Box Collider 2D*) lub kapsuły (*Component*→*Physics 2D*→*Capsule Collider 2D*).
- i. W dodanym komponencie kolizji, dopasuj jego rozmiar (*Radius*) oraz położenie (*Offset*) tak, aby obejmował główny zarys postaci (rys. 6a). [Możesz do tego wykorzystać narzędzie *Edit Collider*](#) .
- j. Uruchom grę – tym razem awatar gracza zatrzyma się na platformie (rys. 6b).

Uwaga

Jeżeli awatar zatrzymuje się nad platformą, to zmniejsz rozmiar obiektu kolizji (lub przesun go w górę), a jeżeli się zapada, to go zmniejsz (lub przesun w dół).

- k. Jeżeli obiekt gracza wciąż przelatuje przez platformę, to dodaj do szablonu platformy komponent kolizji (*Component*→*Physics 2D*→*Box Collider 2D*) i sprawdź ponownie. Skoryguj ewentualne problemy z rozmiarami obiektów kolizji platformy i/lub gracza.

15. Zapisz scenę i projekt. W przypadku zajęć w laboratorium przedstaw efekt prowadzącemu.



Rys. 6. Scena z obiektem gracza oraz platformą: a) okno *Scene* z zaznaczonym komponentem kolizji, b) okno *Game* demonstrujące wykrytą kolizję gracza z platformą.

II. Sterowanie obiektem gracza (20%)

16. W oknie *Project* utwórz folder (*Assets*→*Create*→*Folder* lub →*Folder* lub *PPM*→*Create*→*Folder*) i nazwij go *Scripts*.
17. Zaznacz folder *Scripts* i utwórz w nim skrypt C# (*Assets*→*Create*→*C# Script* lub →*C# Script* w oknie *Project*) i nazwij go *PlayerController*. Skrypt ten będzie odpowiedzialny za zachowanie awatara gracza.
18. Otwórz skrypt w środowisku *MS Visual Studio* (*Open...* w oknie *Inspector* lub klikając dwukrotnie w nazwę skryptu). Upewnij się, że nazwą klasy jest *PlayerController*.

Uwaga

Domyślnie tworzony kod skryptu w języku *C#* zawiera dyrektywy wskazujące na wykorzystywane biblioteki oraz definicję klasy, której nazwa musi być identyczna z nazwą skryptu. Jeżeli tak nie jest, to konieczne zmienić domyślną nazwę klasy *NewBehaviourScript* na nazwę skryptu.

Każda klasa składa się z danych – *pól* (lub mniej formalnie *zmiennych*) i *komponentów* oraz *metod*, które na nich operują i dostarczają określoną funkcjonalność do wykorzystania przez otoczenie danej klasy, a więc przykładowo inne klasy lub silnik *Unity*.

Domyślnie tworzona klasa zawiera definicję dwóch metod:

- *Start()*, która jest wywoływana tuż przed pierwszym wywołaniem metody *Update()* oraz
- *Update()*, która jest wywoływana podczas renderowania każdej ramki gry, a więc na ogół co najmniej 30 razy w ciągu sekundy.

19. Automatyczne przesuwanie awatara w prawo

- a. Zadeklaruj na początku kodu klasy **serializowaną** zmienną **prywatną** *moveSpeed* typu *float* o wartości początkowej *0.1f*, określającej prędkość poruszania się awatara.

```
[Range( 0.01f, 20.0f )] [SerializeField] private float moveSpeed = 0.1f; // moving speed of the player
```

Uwaga

Deklarując pole klasy w środowisku *Unity* należy wziąć pod uwagę trzy aspekty:

1. Czy pole ma być dostępne do edycji w oknie *Inspector*?
2. Czy pole ma być serializowane?
3. Czy pole ma być widoczne dla innych klas?

W przypadku pól, które mają być dostępne w oknie *Inspector* mamy dwie możliwości:

A) Zadeklarowanie zmiennej jako publicznej, np.:


```
public float moveSpeed = 0.1f;
```

B) Zadeklarowanie prywatnej zmiennej serializowanej¹, np.:

```
[SerializeField] private float moveSpeed = 0.1f;
```

Z punktu widzenia *Unity*, obie deklaracje są równoważne, tzn. pola pojawią się w oknie *Inspector* i będą serializowane. Jednak z punktu widzenia paradygmatów programowania obiektowego OOP (ang. *object oriented programming*), zalecane jest drugie z rozwiązań, które zwiększa hermetyczność klasy (ang. *encapsulation*) i to właśnie rozwiązanie będziemy dalej konsekwentnie stosować.

Dodatkowy parametr *Range* pozwala na określenie dla pól liczbowych zakresu, jaki może przyjmować ich wartość² i dodaje w oknie *Inspector* wygodny suwak do ustawiania wartości pola, np.:



¹ <https://docs.unity3d.com/Manual/script-Serialization.html>

² <https://docs.unity3d.com/ScriptReference/RangeAttribute.html>

- b. Dopisz w metodzie `Update()` kod przesuwający obiekt gracza z wykorzystaniem metody `transform.Translate()`³, której parametrami są: wartości przesunięcia w osi X, Y i Z przemnożone przez wartość `Time.deltaTime` oraz parametr `Space.World`, ustalający dla ruchu globalny układ współrzędnych.

```
transform.Translate( moveSpeed * Time.deltaTime, 0.0f, 0.0f, Space.World );
```

Uwaga

Parametr `Time.deltaTime` określa czas jaki upłynął od końca renderowania poprzedniej ramki i może być wykorzystany do uniezależnienia prędkości ruchu od wydajności systemu, na którym gra jest uruchamiana.

Użycie parametru `Space.World` oznacza, że ruch będzie się odbywał w globalnym układzie współrzędnych świata gry, a nie wg lokalnej orientacji obiektu.

- c. Zapisz skrypt (*Ctrl+S*), przejdź do edytora *Unity* i poczekaj na kompilację zmian. W przypadku pojawienia się błędów popraw je w *MS Visual Studio*.
- d. Przeciągnij skrypt *PlayerController* z okna *Project* na obiekt gracza (*Player*) w oknie *Hierarchy*. W oknie *Inspector* dla obiektu gracza powinien pojawić się komponent o nazwie *PlayerController (Script)*, a w nim zadeklarowany parametr *Move Speed* z przypisaną wartością inicjalną.

Uwaga

Ewentualnych zmian wartości zmiennych serializowanych należy dokonywać we właściwościach obiektu w oknie *Inspector*, a nie w kodzie!

- e. Uruchom grę. Awatar gracza powinien przesuwać się w prawo i spaść z prawego końca platformy. Nie przejmuj się, jeżeli na tym etapie prędkość będzie za duża lub za mała.
20. Sterowanie ruchem poziomym awatara za pomocą metody `Input.GetKey()`
- a. Przejdź do edytora *MS Visual Studio* i dopisz w metodzie `Update()` instrukcję warunkową ograniczającą przesuwanie awatara wyłącznie do momentu, w którym gracz wciska klawisz strzałki w prawo (\rightarrow). Do wykrycia wciśnięcia klawisza użyj funkcji `Input.GetKey()`⁴. Klawisz strzałki w prawo zakodowany jest w *Unity* jako `KeyCode.RightArrow`⁵.
- b. Uruchom grę i sprawdź, że można poruszać awatarem używając strzałki w prawo (ale nie z obszaru *NumPad*!). Nie przejmuj się jeżeli na tym etapie prędkość będzie za duża lub za mała.
- c. Dodaj możliwość poruszania awatarem w lewo.
21. Ustalenie prędkości poruszania awatara
- a. Ustalenie właściwej prędkości poruszania się awatara jest możliwa dopiero po kilkukrotnym uruchomieniu i przetestowaniu gry.
- Uruchom grę z **wyłączoną** opcją *Maximize* okna *Game*.
 - Ustal, czy awatar porusza się za szybko, czy za wolno.
 - Zmień parametr *Move Speed* w komponencie *PlayerController* i sprawdź jego wpływ na szybkość poruszania się awatara (może wymagać kliknięcia w okno *Game*).
 - Ustal właściwą wartość i zapamiętaj ją!
 - Wyłącz grę i zauważ, że została przywrócona wartość parametru *Move Speed* sprzed uruchomienia gry.
 - Ustaw ponownie właściwą wartość parametru *Move Speed* i zapisz projekt i scenę.


³ <https://docs.unity3d.com/ScriptReference/Transform.Translate.html>

⁴ <https://docs.unity3d.com/ScriptReference/Input.GetKey.html>

⁵ <https://docs.unity3d.com/ScriptReference/KeyCode.html>

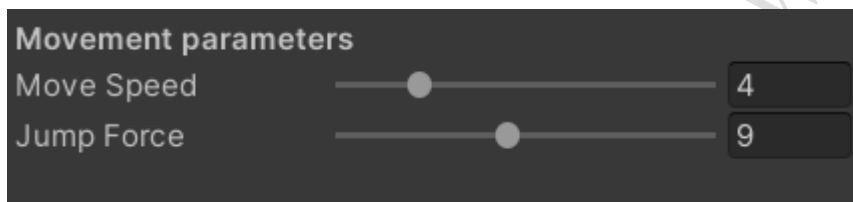
Uwaga

Przywracanie ustawień parametrów sprzed uruchomienia gry jest specyficzną cechą środowiska *Unity*, która pozwala na niezwykle łatwe i (niemal) bezkarne dopasowanie wartości parametrów rozgrywki podczas testowego uruchomienia gry.

W przypadku ustawiania większej liczby parametrów wygodniej jest skorzystać z poleceń kontekstowych skryptu : *Copy Component* w czasie gry oraz *Paste Component Values* po jej zakończeniu.

22. Sterowanie skokiem awatara

- Tuż pod deklaracją pola `moveSpeed`, zadeklaruj **prywatną** zmienną **serializowaną** `float` `jumpForce`, o wartości inicjalnej `6.0f`, określającej wysokość skoku awatara.
- Wiersz przed deklaracją zmiennej `moveSpeed` dodaj atrybut *Header*⁶ [`Header("Movement parameters")`], a w wierszu po deklaracji pola `jumpForce` dodaj atrybut *Space*⁷ [`Space(10)`], co pozwoli wizualnie zgrupować i opisać obydwa parametry w oknie *Inspector* (rys. 7). Jeżeli użycie parametru *Space* powoduje błędy, to usuń lub zakomentuj tę dyrektywę.



Rys. 7. Przykład edytowalnych pól skryptu w oknie *Inspector*.

- Zadeklaruj **prywatne** pole `rigidBody` typu `Rigidbody2D`, które zapewni dostęp do właściwości fizycznych awatara.
- Dodaj metodę `void Awake()` i w jej ciele przypisz do zmiennej `rigidBody` referencję do komponentu `Rigidbody2D` awatara, uzyskaną metodą `GetComponent<Rigidbody2D>()`.

```
rigidBody = GetComponent<Rigidbody2D>();
```

Uwaga

Metoda `Awake()` jest wywoływana w momencie aktywacji obiektu i jest to właściwe miejsce do inicjalizacji zmiennych.

Zaprezentowany mechanizm jest klasycznym sposobem pozyskiwania uchwytu do komponentów obiektów w środowisku *Unity*.

- W metodzie `Update()`, w przypadku wykrycia kliknięcia lewym przyciskiem myszki należy do obiektu awatara przyłożyć siłę skierowaną w górę. Do wykrycia kliknięcia myszką należy użyć metody `Input.GetMouseButtonDown(0)`⁸. Do przyłożenia siły należy użyć metody `AddForce()` komponentu `rigidBody`, której parametrami są: wektor przyłożenia siły (`Vector2.up`) przemnożony przez wartość siły (`jumpForce`) oraz typ przyłożonej siły – tutaj `ForceMode2D.Impulse`.
- Uruchom grę i ustaw właściwą wartość parametru `jumpForce`.

23. W przypadku zajęć w laboratorium przedstaw efekt prowadzącemu.

III. Poprawienie fizyki ruchu awatara (20%)

24. Oznaczenie powierzchni platformy

⁶ <https://docs.unity3d.com/ScriptReference/HeaderAttribute.html>

⁷ <https://docs.unity3d.com/ScriptReference/SpaceAttribute.html>

⁸ <https://docs.unity3d.com/ScriptReference/Input.GetMouseButtonDown.html>

Platforma 2D i ruch awatara

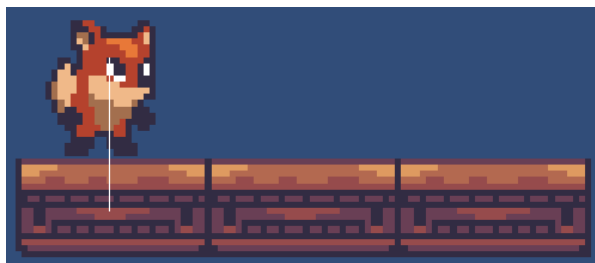
- a. Zadeklaruj w skrypcie gracza *PlayerController* zmienną publiczną *groundLayer* typu *LayerMask*.
- b. Przejdź do *Unity*, otwórz okno edycji etykiet i warstw (*Edit*→*Project Settings*→*Tags and Layers* lub w oknie *Inspector Layers*→*Edit Layers*) i dodaj warstwę o nazwie *Ground* jako pierwszą warstwę użytkownika (*User Layer 6*).
- c. Ustaw warstwę *Ground* we właściwości *Layer* wszystkich wykorzystywanych szablonów platform na scenie, jak również we wszystkich zasobach platform (*Assets*→*Prefabs* w oknie *Project*), których możesz w przyszłości używać jako platformy w projekcie.

Uwaga

Jeżeli masz problem ze znalezieniem szablonu (ang. *prefab*) użytego do stworzenia danego obiektu na scenie, możesz użyć opcji *Select Prefab* z jego menu kontekstowego w oknie *Hierarchy*.

- d. Zaznacz obiekt gracza i ustaw w oknie *Inspector* własność *Ground Layer* w komponencie *PlayerController (Script)*→*Ground Layer* (ale NIE zmieniaj właściwości *Layer* obiektu *Player*).
 - e. Zapisz scenę i projekt (*wyrób sobie nawyk*) i przejdź do edytora *MS Visual Studio*.
25. Sprawdzenie, czy gracz dotyka platformy
- a. Zadeklaruj stałą *rayLength* typu *float* o wartości 2.0f;
 - b. Dodaj na końcu funkcji *Update()* wywołanie funkcji *Debug.DrawRay()*, której parametrami są: położenie obiektu (*transform.position*), długość wektora w dół (*rayLength * Vector3.down*), kolor kreski (np. *Color.white*), czas wyświetlania w [s] (np. 1) oraz możliwość uwzględnienia przesłaniania przez inne obiekty (tu *false*).
 - c. Uruchom program i zaobserwuj linię prowadzącą od środka obiektu gracza w dół. Jeżeli nie widzisz linii, kliknij na opcję *Gizmos* w oknie *Game*;
 - d. Dopasuj długość linii tak, aby sięgała mniej więcej do połowy wysokości platformy (rys. 8).
 - e. Zdefiniuj nową metodę *IsGrounded()*, która zwraca informację (typu logicznego *bool*), czy awatar znajduje się „tuż” nad powierzchnią platformy. Jednym z możliwych sposobów jest użycie metody *Physics2D.Raycast()*⁹, która rzuca w zadanym kierunku promień o zadanej długości i sprawdza, czy przeciął on kształt kolizji (ang. *collider*) dowolnego obiektu. Użyj wersji metody, która umożliwia ograniczenie sprawdzania kolizji do obiektów przynależących do zadanej warstwy. Parametrami metody są kolejno: aktualna pozycja obiektu, wektor jednostkowy skierowany w dół, zasięg promienia (*rayLength*) oraz maska warstwy (*groundLayer.value*).

```
bool IsGrounded()
{
    return Physics2D.Raycast(this.transform.position, Vector2.down, rayLength, groundLayer.value);
}
```



Rys. 8. Przykładowa wizualizacja długości promienia metodą *Debug.DrawRay()*.

26. Utworzenie metody odpowiedzialnej za skoki
- a. Utwórz metodę *Jump()*, która po sprawdzeniu, czy awatar znajduje się na platformie (*IsGrounded()*), pozwala mu podskoczyć. Wykorzystaj metodę *rigidBody.AddForce()* użytą już wcześniej w metodzie *Update()*.

⁹ <https://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html>

- b. Zmodyfikuj metodę `Update()` tak, aby po naciśnięciu lewego przycisku myszki wykonywana była metoda `Jump()`.
- c. Sprawdź poprawne działanie skoków, wyłącznie w momencie, w którym awatar stoi na platformie.

Uwaga

Gdy nabierzesz wprawy w środowisku *Unity*, możesz niemal dowolnie zmieniać sterowanie graczem, dodając np. możliwość dodatkowego odbicia się w powietrzu, oddania dłuższego skoku przy wciśniętym klawiszu funkcyjnym, używanie wyłącznie sił przykładanych przez gracza i sił tarcia itp.).

- d. Dodaj w metodzie `Jump()` wyświetlenie komunikatu w konsoli metodą `Debug.Log()`, której parametrem może być łańcuch znaków (np. "jumping"). **Zaobserwuj pojawiający się w konsoli komunikat, przy każdorazowym podskoku awatara.**
- e. Rozbuduj warunki odpowiedzialne za poruszanie awatarem tak, aby możliwe było ono również poprzez użycia klawisza *A* (`KeyCode.A`) i *D* (`KeyCode.D`).
- f. Rozbuduj warunek odpowiedzialny za skok tak, aby był on możliwy również poprzez użycia klawisza spacji (`KeyCode.Space`). Użyj tym razem funkcji `Input.GetKeyDown()`.

Uwaga

Funkcja `Input.GetKey()` symuluje do pewnego stopnia sterowanie analogowe i uwzględnia autopowtarzanie, tzn. może zostać wywołana kilka razy mimo jednorazowego przyciśnięcia klawisza. O ile nie stanowi to kłopotu w przypadku sterowania ruchem na boki, to może być kłopotliwe w przypadku skoków!

Funkcja `Input.GetKeyDown()` wywoływana jest wyłącznie raz po naciśnięciu danego klawisza, niezależnie od długości jego przyciśnięcia.

- g. Sprawdź poprawność działania sterowania awatarem gracza. W razie potrzeby skoryguj parametry odpowiedzialne za wysokość skoku oraz szybkość jego poruszania się po planszy.

Uwaga

Bardziej uniwersalną metodą sterowania jest użycie tzw. osi¹⁰ (ang. *axis*), które pozwala łączyć różne metody sterowania imitując m.in. działanie analogowego dżojstika (ang. *joystick*).

27. Może się zdarzyć, że przy dużej prędkości poruszania się (np. spadania) awatara, silnik fizyki nie wykryje momentu kolizji z platformą i obiekt gracza przeleci przez nią. Aby tego uniknąć, zmień właściwość obiektu gracza *Rigidbody 2D* → *Collision Detection* z *Discrete* na *Continuous*.

Uwaga

Nie należy używać dokładnego sprawdzania kolizji dla wolno poruszających się lub nieruchomych obiektów, gdyż może to znacząco zwiększyć wymagania obliczeniowe!

28. W przypadku zajęć w laboratorium przedstaw prowadzącemu efekty prac. Włącz wyświetlanie promienia i pokaż, że lisek nie może odbijać się w trakcie skoku.

¹⁰ <https://docs.unity3d.com/ScriptReference/Input.GetAxis.html>

IV. Mapa kafelków (20%)

Uwaga


Wiele pakietów grafik typu rastrowego (ang. *pixel art*), np. *Sunny Land*, zawiera zestaw tzw. kafelków (ang. *tile set*) w formie atlasu tekstur (ang. *texture atlas*). W przypadku obecności takiego atlasu istnieje możliwość łatwego ‘malowania’ poziomów gry z wykorzystaniem mechanizmów oferowanych przez edytor *Unity*.

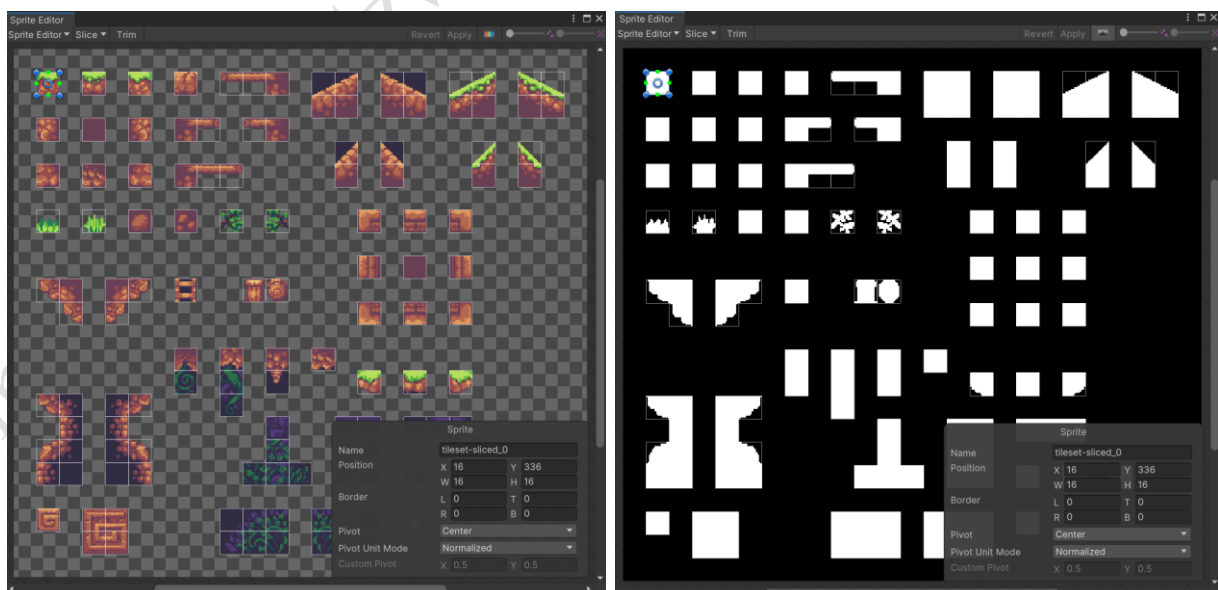
29. Przygotowanie zestawu kafelków (dla pakietu *SunnyLand* realizacja tego punktu nie jest wymagana, gdyż zawarty w nim zestaw kafelków został już odpowiednio przygotowany).

- Odnajdź w pakiecie zasobów zestaw kafelków (*SunnyLand Artwork*→*Environment*→*tileset-sliced*).
- W oknie *Inspector* ustaw odpowiednie parametry:
 - Texture Type = Sprite (2D and UI),
 - Sprite Mode = Multiple,
 - Pixels Per Unit* = 16 (bazowa szerokość kafelków, może być inna dla innych pakietów),
 - Filter Mode = Point (no filter),
 - Compression = None.
- W przypadku wprowadzenia zmian, zatwierdź je przyciskiem *Apply*.
- Otwórz okno edytora kafelków (*Window*→*2D*→*Sprite Editor*) i je zmaksymalizuj. Wybierz opcję menu *Slice*, ustaw sposób podziału *Type* na *Grid By Cell Size* z parametrami *Pixel Size* = (16,16) i wydziel kafelki przyciskiem *Slice* (rys. 9).

Uwaga

Opcja podziału *Automatic* wykrywa większe obiekty składające się z kilku kafelków i traktuje je jako całość, ale ogranicza możliwość wykorzystania mniejszych fragmentów.

- Zaobserwuj ramki dookoła poszczególnych grafik, a po wybraniu dowolnej z nich - okienko z jej ustawieniami. Zatwierdź ustawienia opcją *Apply*. Możesz zaobserwować maski (kanał przejrzystości) utworzonych kafelków korzystając z małego przycisku  na prawo od opcji *Apply* (rys. 9).




Rys. 9. Okno *Sprite Editor* z zaznaczonym podziałem grafiki (po lewej) oraz wyznaczone maski nieprzezroczystości (po prawej).

30. Przygotowanie map kafelków (ang. *tile map*)

- a. Dodaj do sceny obiekt mapy kafelków *GameObject*→*2D Object*→*Tilemap*→*Rectangular*. Powstały obiekt *Grid* stanowi podstawę do nakładania i wyświetlania map kafelków, składających się z jednej lub większej liczby warstw kafelków *Tilemap*.
- b. Wskaż w oknie *Hierarchy* obiekt *Tilemap* i zmień jego nazwę na *Platforms*.

Uwaga

W przypadku, gdy rozmiar siatki nie odpowiada wielkość obiektów na scenie, należy upewnić się, że zaimportowane obiekty mają właściwą rozdzielczość (16 dla *SunnyLand*). Jeżeli ich rozdzielczość jest prawidłowa, to zmienić skalę obiektów albo siatki - od razu dla wszystkich warstw bezpośrednio w obiekcie *Grid* (*Grid*→*Transform*→*Scale XY*) lub alternatywnie *Grid*→*Cell Size XY* - albo dla poszczególnych warstw. W obydwu przypadkach warto zablokować proporcje siatki ( *Transform*→*Scale*).

- c. Otwórz okno palety kafelków (*Window*→*2D*→*Tile Palette*) i nieco je powiększ. Na większych monitorach możesz je zadokować jako nowy panel obok panelu zawierającego okno *Inspector* (nie w tym samym panelu).
- d. W oknie powinna już być uaktywniona paleta *Main Palette*, która jest zawarta w pakiecie *SunnyLand*.

Uwaga

W celu utworzenia nowej palety, należy ją utworzyć opcją *Create New Palette* z rozwijanej listy, odpowiednio ją nazwać (np. *Sunny Land Palette*), zaakceptować domyślne parametry i zapisać w odpowiednim folderze, np. *Assets*→*Tile Palettes*. Następnie należy przeciągnąć zestaw kafelków na obszar okna palety i zapisać je w odpowiednim folderze, np. *Assets*→*Tiles*.

31. Malowanie kafelkami



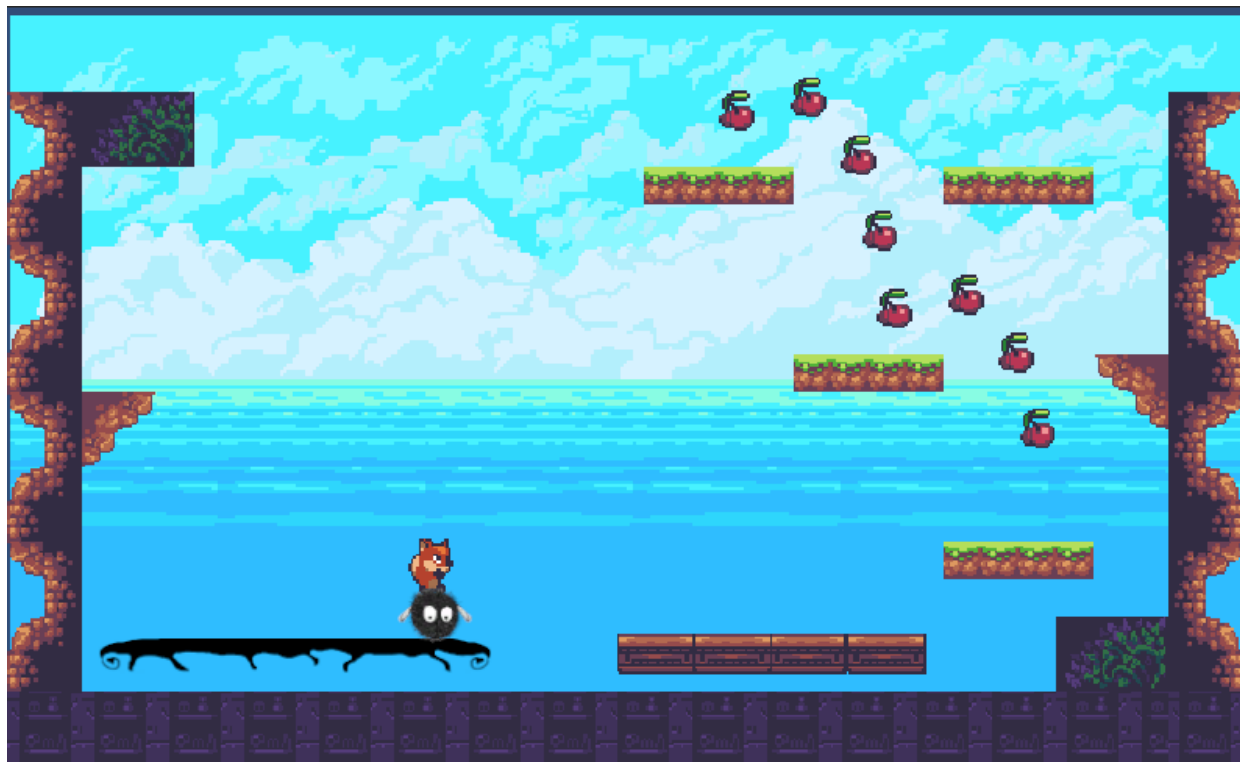
- a. Okno palety zawiera zestaw narzędzi umożliwiających łatwe tworzenie poziomów:
 - i. Zaznaczanie obszaru na siatce w oknie sceny;
 - ii. Przesuwanie zaznaczonego obszaru (wraz z zawartością);
 - iii. Malowanie aktualnie wybranym elementem palety. Można zaznaczyć dowolny obszar palety! Z wciśniętym klawiszem *Shift* można wymazywać zawartość mapy kafelków.
 - iv. Zamalowanie wybranym kafelkiem zaznaczonego prostokątnego obszaru (ang. *marquee selection*);
 - v. Wybranie lub wskazanie obszaru palety jako nowego pędzla (ang. *marquee selection*); Opcja zbędna, gdyż działa również przy malowaniu pędzlem (iii).
 - vi. Wymazanie kafelków z mapy; Opcja również zbędna, gdyż działa identycznie do malowania z wciśniętym klawiszem *Shift*.
 - vii. Wypełnienie zamkniętego obszaru sceny aktualnie wybranym pędzlem z palety.

Uwaga

Istnieją dodatkowe narzędzia związane z „malowaniem” kafelkami. Można je udostępnić w opcji preferencji *Edit*→*Preferences*→*2D*→*Tile Palette*→*Default Tile Palette Tools*. Umożliwiają one m.in. zalewanie obszaru (ang. *floodfill*), lustrzane odbicie kafelka (ang. *flip X/Y*) oraz obrót kafelka o $\pm 90^\circ$.

Jeżeli rozmiar kafelków nie zgadza się z rozmiarem siatki, to oznacza, że został popełniony błąd na którymś z poprzednich etapów. Tymczasowo można dostosować rozmiar komórek obiektu *Grid* (por.30.b) jednak w docelowej grze należy to powtórnie wykonać tym razem prawidłowo.

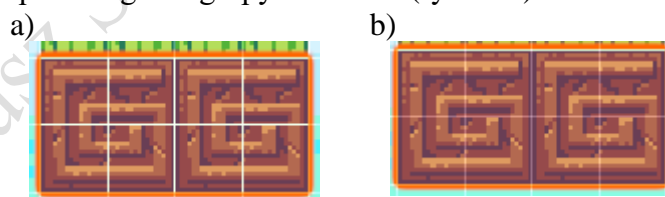
- b. Zwiększ pole widzenia kamery i narysuj z wykorzystaniem pędzla i innych narzędzi kilka dodatkowych platform (zrób to lepiej niż na rys. 10 😊).



Rys. 10. Przykładowa scena prezentująca połączenie dotychczasowych elementów i kafelków.

32. Ustawienie kolizji dla kafelków

- Dla obiektu *Grid*→*Platforms* ustaw parametr *Layer* na *Ground* i dodaj dedykowany komponent kolizji (*Tilemap*→*Tilemap Collider 2D*). **Zauważ pojawienie się komponentu kolizji dla każdego kafelka w tej warstwie (rys. 11a).**
- Ustaw gracza nad warstwą kafelków i sprawdź, że może poruszać się po nich podobnie jak po platformie.
- Ustawienie komponentu kolizji dla każdego kafelka może powodować niepotrzebne obciążenie systemu fizyki sprawdzających wewnętrzne kolizje z krawędziami pomiędzy kafelkami. Aby tego uniknąć, dodaj do obiektu *Grid*→*Platforms* komponent *Composite Collider 2D*. Uruchom grę i zauważ spadanie kafelków pod wpływem automatycznie dodanego komponentu *Rigid Body 2D*. Aby tego uniknąć zmień typ ciała *Body Type* z *Dynamic* na *Static*.
- W komponencie *Tilemap Collider 2D* zaznacz właściwość *Used By Composite*, aby zmienić obiekty kolizji na otaczające poszczególne grupy kafelków (rys. 11b).



Rys. 11. Fragment 8 kafelków z widocznymi krawędziami kolizji również wewnątrz grupy (a) oraz jedynie na brzegach (b).

- Zaznacz obiekt *Grid*, utwórz dodatkową warstwę kafelków (*GameObject*→*2D Object*→*Tilemap*→*Rectangular*) o nazwie *Props*, a następnie ustaw jej odpowiednią skalę i odległość od kamery (*Position*→*Z*). Do warstwy *Props* dodaj z palety elementy, które nie mają pełnić roli platformy (**gracz nie ma się od nich odbijać**).


34. Dodanie tła

- Znajdź dowolną grafikę na tło sceny, np. *SunnyLand Artwork*→*Environment*→*back*.
- Ustaw właściwą skalę obrazka (tu *Pixels Per Unit*=16), a następnie przeciągnij obrazek na scenę. **Jeżeli wciąż aktywne są elementy *Tilemap* i *Tile Palette*, to zrezygnuj z dodania obrazka do siatki/palety i przeciągnij go na nazwę poziomu.**
- Przeskaluj obrazek tak, aby objął tło utworzonej sceny (rys. 10).

35. Podział na warstwy

Uwaga

Aby uniknąć problemów z kolejnością renderowania na scenie obiektów znajdujących się w tej samej odległości od kamery (ang. *z-fighting*) można użyć dwóch podejść – zmieniać odległość współrzędnej Z obiektów (im większa warstwa tym dalej od kamery) lub zastosować warstwy renderowania i określić ich kolejność. Pierwsze podejście jest nieco prostsze i sprawdza się przy niewielkiej liczbie warstw, jednak wykorzystanie warstw pozwala na większą elastyczność.

- a. Wybierz z paska narzędziowego opcję *Layers*→*Edit Layers* .. () rozwiń panel *Sorting Layers* i dodaj (+) warstwy: *Background*, *Tilemap*, *Props*, *Platforms*, *Player* and *Enemies*, *Foreground* (rys. 12).

Uwaga

Warstwy wyświetlane są w kolejności od najwyższego numeru (*Layer 6*). W przyszłości możesz dodać kolejne warstwy, jak również zmieniać ich kolejność poprzez przeciąganie warstw w hierarchii.

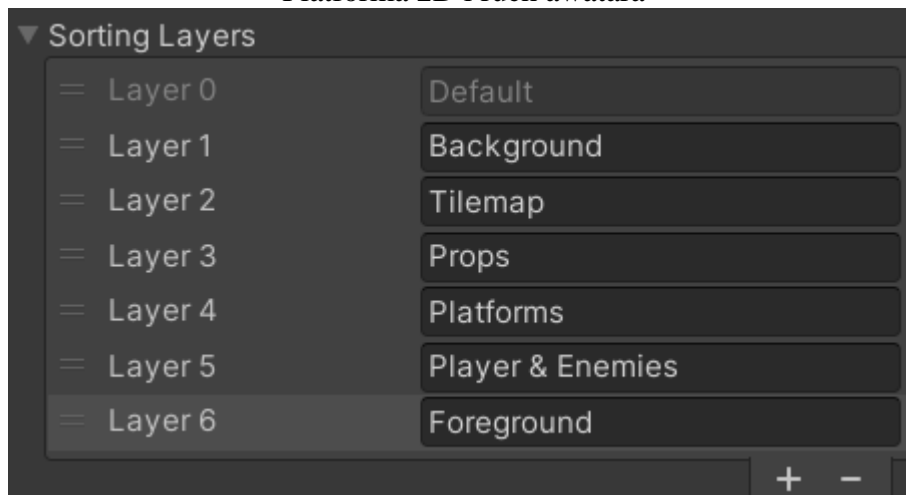
Warstwa *Default* może być traktowana jako ostatnia warstwa, co wymusza najczęściej konieczność przypisywania innej warstwy każdemu nowo dodanemu obiektowi. Można też ustawić ją jako jedną z warstw środkowych w hierarchii, co będzie wygodne przy prototypowaniu poziomu.

- b. Zaznacz w oknie *Hierarchy* obiekt *Grid*→*Props* i w oknie *Inspector* w komponencie *Tilemap Renderer* ustaw właściwość *Additional Settings*→*Sorting Layer* na *Props*.
- c. Analogicznie ustaw tę właściwość w obiekcie *Grid*→*Platforms* na *Tilemap*.
- d. Analogicznie dla obiektu *Player* w komponencie *Sprite Renderer* ustaw tę właściwość na *Player and Enemies*.
- e. Analogicznie dla obiektu *Background* ustaw tę właściwość na *Background*.
- f. Sprawdź, czy wszystkie obiekty są widoczne. W przypadku nieprawidłowej widoczności ustaw w obiekcie odpowiednią kolejność renderowania lub popraw kolejność sortowania warstw.

Uwaga

Pamiętaj, żeby ustawić odpowiednią warstwę dla obiektów dodawanych w późniejszych lekcjach, tak aby były widoczne na scenie!

36. Sprawdź działanie gry – w razie potrzeby powiększ obszar obejmowany przez kamerę. W przypadku zajęć w laboratorium przedstaw efekt prowadzącemu.



Rys. 12. Przykładowy zestaw warstw renderowania w kolejności od *Foreground* (zawsze na wierzchu) do *Default* (ostatnia warstwa).

V. Tuning (10%)

37. Dodaj do poziomu grafiki oznaczające punkt startowy i końcowy
38. Dla punktu końcowego dodaj detekcję kolizji z graczem i wyświetlanie w konsoli komunikatu „Game over”. Do wykrywania kolizji wykorzystaj metodę `void OnCollisionEnter(Collision col)` umieszczoną w skrypcie obiektu końcowego poziomu. Jeżeli obiekt nie ma blokować przejścia gracza, to należy ustawić we właściwościach jego *Collidera* opcję *isTrigger* i wykorzystać metodę `void OnTriggerEnter(Collider col)`.
39. W przypadku zajęć w laboratorium przedstaw efekt prowadzącemu.

VI. Praca domowa – wstępny projekt poziomu gry

40. Jeżeli nie udało się zrealizować całej instrukcji w trakcie zajęć, to należy ją dokończyć przed kolejnymi zajęciami, gdyż wszystkie dotychczasowe elementy będą wykorzystywane na kolejnych zajęciach.
41. W oparciu udostępnione przykłady gier z poprzednich lat opracuj w zespole wstępną koncepcję poziomu biorąc pod uwagę elementy, które zostaną zrealizowane na kolejnych zajęciach, w tym m.in.:
 - a. platformy, w tym ruchome, poruszające się po prostych trasach poziomych, pionowych (winda) i skośnych, a także po cyklicznych trasach złożonych (wiele punktów kontrolnych);
 - b. obiekty składające się z wielu synchronicznie poruszających się platform, np. karuzela-młyn;
 - c. przeciwników patrolujących określone platformy i/lub przestrzeń pomiędzy;
 - d. bonusy (pop. znajdźki) zwiększające wynik końcowy gry;
 - e. 3 klucze, które muszą być zebrane, aby ukończyć poziom;
 - f. inne elementy, np. portale, ukryte przejścia, drabiny, liany, obiekty przesuwane itp.
42. Wykonaj tuning poziomu zawierający dotychczasowe elementy oraz miejsce na wykonanie kolejnych, a więc m.in. przestrzeń, której nie da się przeskoczyć bez ruchomej platformy, z miejscem na przeciwników, klucze i bonusy itp. Zapewnij, aby od startu do mety była możliwość przejścia co najmniej dwiema alternatywnymi drogami.
43. Postaraj się, aby poziom był w miarę estetyczny.

UWAGA! Po zakończeniu zajęć WYLOGUJ SIĘ, zamknij edytor Unity, usuń z folderu projektu podfolder **Library**. Następnie skompresuj folder projektu do formatu ZIP i wgraj go na dowolny nośnik lub dysk sieciowy. Będzie on potrzebny na kolejnych zajęciach!