Piszemy RPGo-Platformówkę (3) – Wyginaj śmiało ciało! (sheadovas/poradniki/proj_platf_rpg/3-wyginaj-smialo-cialo/)

Mar 26, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Dorabiamy wielokrotne skoki oraz „bieganie”

Hej, dzisiejszy materiał jest poniekąd uzupełnieniem [Nauka chodzenia (sheadovas/poradniki/proj_platf_rpg/1-nauka-chodzenia/)] i omawiany kod bazuje na commicie [2697751 (https://github.com/sheadovas/proj_platf_rpg/commit/26977512328e359d0d33fde484e8b71708618d35)].

Dzisiaj umożliwimy graczowi skakanie oraz bieganie, w dalszych materiałach stworzymy sobie także odbijanie od ścian. Miłego czytania.

Wyginaj śmiało ciało!

Wielokrotne skoki

Zacznijmy od skoków, które będziemy mogli wykonać wiele razy. Sam mechanizm jest dość prosty do zaimplementowania, bo jest poniekąd zmianą warunku na umożliwienie pojedynczych skoków.

> Ciekawostka: do tej pory gracz miał nieograniczoną liczbę skoków, więc mógł „latać”.

Ogólny algorytm takiego kodu może wyglądać następująco:

1. Ustaw maksymalną ilość skoków, oraz ustaw na tą samą wartość ilość dostępnych skoków.
2. Gdy gracz będzie chciał skoczyć: sprawdź czy ilość dostępnych skoków jest większa od 0:
    1. jeżeli jest większa: wykonaj skok i zmniejsz ilość dostępnych skoków o 1;
    2. w przeciwnym wypadku: nic nie rób.
3. Jeżeli gracz dotknie ziemi to przywrócić mu maksymalną ilość skoków.

Jak widzimy, to jedyny problem może nam sprawić przywrócenie bazowej ilości skoków gdy gracz opadnie na ziemię (tzn. problemem jest wykrycie czy gracz jest na ziemi), ale tym zajmiemy się później.

PlayableCharacter.cs C#

```
...
public int jumpLimit = 1;
protected int m_availableJumps;
...
```

Jak widzimy na kodzie powyżej doszły nam dwie zmienne, domyślnie postać może wykonać pojedynczy skok (*jumpLimit*), druga zmienna (*m_availableJumps*) mówi ile jeszcze graczowi pozostało skoków.

PlayableCharacter.cs C#

```
  protected virtual void Jump()
  {
    if (m_controller.isJumpClicked && m_availableJumps > 0)
    {
      m_availableJumps--;
      m_rigidbody.velocity = new Vector2(m_rigidbody.velocity.x, 0.0f); // reset falling speed
      m_rigidbody.AddForce(new Vector2(0, m_hspeed * mass));
    }
  }
```

Kolejny listing do zmieniona funkcja *Jump*, która sprawdza czy gracz może wykonać skok, jeżeli tak to postępuje zgodnie z opisanym wcześniej algorytmem. Warty uwagi jest fakt, że resetujemy obecnie działające siły (grawitacji) na postać, tak aby skok faktycznie się odbył, a nie tylko np. zminimalizował prędkość upadku.

PlayableCharacter.cs C#

```
203   protected void OnGroundEnter(Collider2D other)
204   {
205     m_availableJumps = jumpLimit;
206   }
```

Tutaj mamy funkcję, którą wywołujemy w momencie gracz upadnie na ziemię. W moim przypadku zdecydowałem się stworzyć dodatkowy collider na graczu (z zaznaczoną opcją *isTrigger*), który w momencie zajścia wywoła mi powyższą metodę.

Co więcej zrealizowałem to na osobnym obiekcie, przez co stworzyłem klasę przekierowującą informacje o zdarzeniach tego typu poprzez wywołanie konkretnej funkcji. Prezentuje się ona następująco:

CollisionReceiver.cs C#

```
1  using UnityEngine;
2
3  public class CollisionReceiver : MonoBehaviour
4  {
5    public delegate void CallbackCollisionEnter(Collision2D coll);
6    public delegate void CallbackTriggerEnter(Collider2D coll);
7
8    public CallbackCollisionEnter onCollisionEnterCallbacks;
9    public CallbackTriggerEnter onTriggerEnterCallbacks;
10
11   private void OnCollisionEnter2D(Collision2D collision)
12   {
13     if (onCollisionEnterCallbacks != null)
14       onCollisionEnterCallbacks(collision);
15   }
16
17   private void OnTriggerEnter2D(Collider2D collision)
18   {
19     if (onTriggerEnterCallbacks != null)
20       onTriggerEnterCallbacks(collision);
21   }
22 }
```

Jak widać, ta klasa to po prostu tablica delegatów uruchamianych na zajście kolizji. Ta klasa przyda nam się także później.

PlayableCharacter.cs C#

```
1   protected virtual void Start()
2   {
3     m_availableJumps = jumpLimit;
4     m_groundCheckReceiver.onTriggerEnterCallbacks += OnGroundEnter;
5   }
```

Powyżej widzimy początkowe ustawienie limitu skoków oraz funkcji resetującej ilość dostępnych skoków na kolizję z ziemią.

Jedynie co teraz należy zrobić, to zmienić limit dostępnych skoków, np na ‚2', aby cieszyć się wielokrotnymi skokami.

Bieganie

Można to zrealizować na wiele sposobów, w jednym z najprostszych scenariuszy bieganie to zwiększenie k-razy bazowej prędkości, tak my też to zrealizujemy.

W tym celu musimy zmienić interfejs dla kontrolera, tak aby obsługiwał dodatkowy klawisz odpowiedzialny za skoki:

ICharacterController.cs C#

```
public interface ICharacterController
{
  string controllerType { get; }

  float moveDirection { get; }
  bool isJumpClicked { get; }
  bool isRunningKeyClicked { get; }

  void Control();
}
```

Sama implementacja jest trywialnie prosta:

ManualKeyboardController.cs C#

```
  ...
  public KeyCode keyRun = KeyCode.LeftShift;
  ...
  public void Control()
  {
    ...

    /* Running */
    if(Input.GetKeyDown(keyRun))
    {
      m_isRunning = true;
    }
    else if(Input.GetKeyUp(keyRun))
    {
      m_isRunning = false;
    }
  }
```

W naszej grze żadna ze sztucznej inteligencji nie będzie potrafiła biegać, przynajmniej nie w ten sposób. Dlatego samo obsłużenie tego kodu odbędzie się w klasie *Player*:

Player.cs C#

```
using UnityEngine;

public class Player : PlayableCharacter
{
  public float runningSpeedMultiplier = 2.0f;

  protected override void Move()
  {
    float vy = m_rigidbody.velocity.y;
    float vx = m_vspeed * m_controller.moveDirection;

    if (m_controller.isRunningKeyClicked)
      vx *= runningSpeedMultiplier;

    m_rigidbody.velocity = new Vector2(vx, vy);
  }
}
```

Powyższa implementacja dodaje nowy parametr, decydujący o stopniu przyspieszenia, oprócz tego musieliśmy zmodyfikować metodę *Move*, która przy wciśnięciu przez gracza klawisza biegu (w naszym przypadku jest to lewy Shift) dodatkowo zwiększy prędkość gracza.

diff

W tym paragrafie chciałbym zwrócić uwagę na małą zmianę dotyczącą implementacji Unity. Mianowicie poprzednia implementacja sterowania przy użyciu klawiatury ma małe problemy i się lekko przycina. Obecnie próbuję znaleźć przyczynę i na ten czas aby nie blokować kolejnych części postanowiłem używać funkcji wbudowanej w Unity.

ManualKeyboardController.cs C#

```
57  public void Control()
58  {
59    // HACK Temporary change, fix later
60    m_movdir = Input.GetAxis("Horizontal");
61    ...
62  }
```

W momencie gdy uda mi się naprawić przyczynę tego problemu to w tym miejscu pojawi się odpowiednia notka i link do commita poprawiającego ten problem. To tak, żeby nie było że kogoś gdzieś oszukuję ;)

Podsumowanie

Dzisiaj wzbogaciliśmy grę o 2 nowe funkcjonalności oraz o 1 nową klasę, którą zaczniemy stosować szerzej w przyszłości.

To tyle jeżeli chodzi o dzisiejszy wpis, tradycyjnie zapraszam do systemu komentarzy, polubienia strony na fb, a także do „zagrania” ([wersja z tej lekcji (https://github.com/sheadovas/proj_platf_rpg/releases/tag/1.3)]).

Do przeczytania w kolejnej części, w której zaimplementujemy sobie prosty system walki.

Code ON!