Piszemy RPGo-Platformówkę (11) – [Masz] i Moją tarczę! (sheadovas/poradniki/proj_platf_rpg/11-masz-i-moja-tarcze/)

Lip 12, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Druga część wprowadzająca ekwipunek do gry

(...) i nie ostatnia!

W tym wpisie zajmiemy się dodaniem klasy przedmiotów pośrednich, które niby są przedmiotami, ale z drugiej nimi nie są (o tym dokładniej za chwilę). Zanim przejdziemy dalej to jeszcze zanim pojawi się w Waszych głowach myśl „Ile można ciągnąć temat ekwipunku?", to usprawiedliwię się: logika „przedmiotów pośrednich" jest zaimplementowana, kolejna część zaprezentuje obsługę przedmiotów z poziomu UI (pozostanie nam do roboty czyste UI).

Tradycyjnie zapraszam do wcześniejszego zapoznania się z omawianymi zmianami: [diff (https://github.com/sheadovas/proj_platf_rpg/compare/8adb363b8bb472fed4be224e33e9a3a740c8f522...6e82abb358423498c00543388fbc6be7af0e8ef5)], oraz do zagrania w najnowsze [demo (https://github.com/sheadovas/proj_platf_rpg/releases/tag/1.8)].

Teoria

Wyjątkowo część teoretyczna będzie krótka.

Potrzebujemy specjalnego przedmiotu (uwaga: masło maślane), który będzie reprezentantem Przedmiotów (obiektów dziedziczących po *Item*) na scenie gry. Jak zapewne pamiętacie, to implementacja z poprzedniej wersji zajmowała się jedynie reprezentacją przedmiotu w menu ekwipunku (musimy zająć się tym brakiem).

(...) i Moją tarczę!

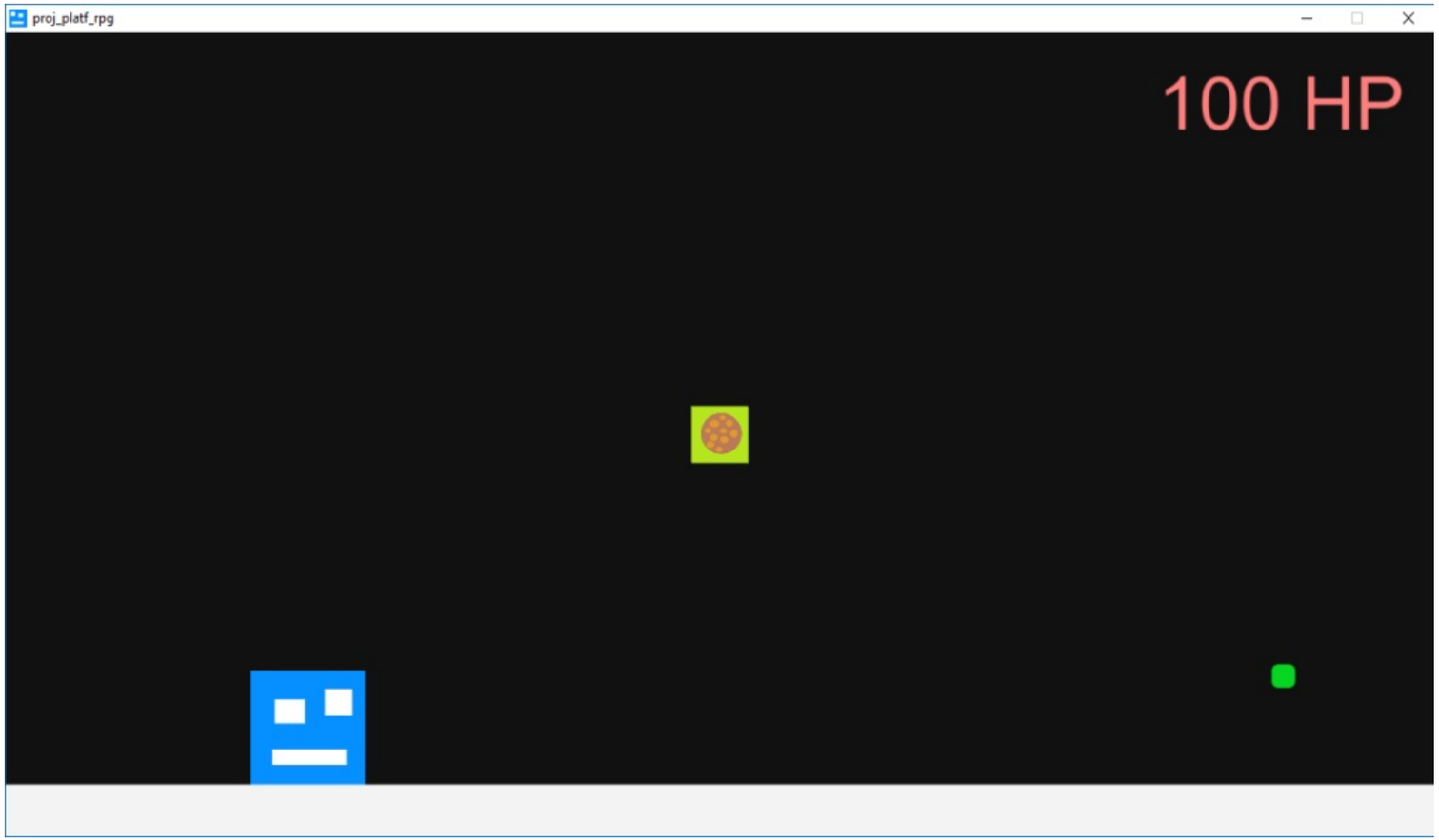
Sam wrapper (pojemnik jeżeli wolicie po polsku) na klasę *Item* jest mega prosty:

ItemObject.cs C#

```
using UnityEngine;

// Wrapper on real object, needed only for visibility in scene
public class ItemObject : MonoBehaviour
{
  public Item item; // real item object

  private CircleCollider2D _triggerCollider;

  [SerializeField]
  private SpriteRenderer _renderer;

  public void SetVisibleOnScene(bool visible, Vector3 position)
  {
    _triggerCollider.enabled = visible;
    _renderer.enabled = visible;

    transform.position = position;
  }

  protected virtual void OnTriggerEnter2D(Collider2D collision)
  {
    item.OnItemTrigger(collision);
  }

  protected virtual void OnCollisionEnter2D(Collision2D collision)
  {
    item.OnItemCollide(collision);
  }

  private void Awake()
  {
    _triggerCollider = GetComponent<CircleCollider2D>();
```

Nie robi praktycznie nic, służy jedynie do graficznej reprezentacji przedmiotów, które leżą luzem (ważne!) i np. są do zebrania, w przypadku eventów przekazuje je do obiektu-rodzica (obiektu, który reprezentuje), gdzie dane zdarzenie może zostać obsłużone.

Oprócz tego potrafi pojawić się na scenie (sytuacja gdy możemy go zebrać) lub zniknąć (gdy zostanie zebrany), służy do tego metoda *SetVisibleOnScene* włączająca/wyłączająca renderer oraz collider.



(https://i2.wp.com/szymonsiarkiewicz.pl/wp-content/uploads/2017/07/rgp-itemobject-beforecollect.png)

Przed zebraniem przemiotu

Nie obyło się też bez dodatkowych modyfikacji w innych źródłach:

PlayableCharacter.cs C#

```
181   protected virtual void OnTriggerEnter2D(Collider2D collision)
182   {
183     ItemObject itemObject = collision.gameObject.GetComponent<ItemObject>();
184     if(itemObject != null)
185     {
186       // we are here only during collison with item
187       // we are hitting it (ItemObject) -> item is collectable
188       CollectItem(itemObject.item);
189     }
190   }
```

W *PlayableCharacter* zaczęliśmy obsługiwać zdarzenie kolizji i w momencie gdy następuje ona z *ItemObject* domyślnie zbieramy ten przedmiot:

PlayableCharacter.cs C#

```
  protected void CollectItem(Item item)
  {
    Debug.Log("Collect item!");

    item.SetPhysicalOnScene(false, item.transform.position);
    equipment.AddItem(item);
  }
}
```

A więc powodujemy, że obiekt zniknie (ważna uwaga: wywołujemy metodę z klasy *Item*, a nie *Item Object!*) oraz dodajemy go do ekwipunku gracza.

Największa ilość zmian przypadła wewnątrz klasy *Item*, pierwszą zmianą jest oczywiście dodanie parametrów umożliwiających odwoływanie się do reprezentanta obiektu oraz umożliwiwających jego stworzenie:

Item.cs C#

```
  [Header("Physical Item Representation")]
  // Representation of item in gameplay (non-ui) scene
  [SerializeField]
  protected ItemObject m_itemObject;

  // Create physical representation of object during init of this item
  [SerializeField]
  private bool _physicalOnInit = false;

  // Option for above option
  // Initialize object on position
  [SerializeField]
  private Vector3 _physicalInitPosition = Vector3.zero;

  // Required to instastate item representation (prefab of object)
  [SerializeField]
  private GameObject _physicalObjectPrefab;
  #endregion

  #region Public methods
  public virtual void OnItemCollide(Collision2D collision)
  {

  }

  public virtual void OnItemTrigger(Collider2D collider)
  {

  }
```

Mamy też metodę, której wywołanie widzieliśmy powyżej:

Item.cs C#

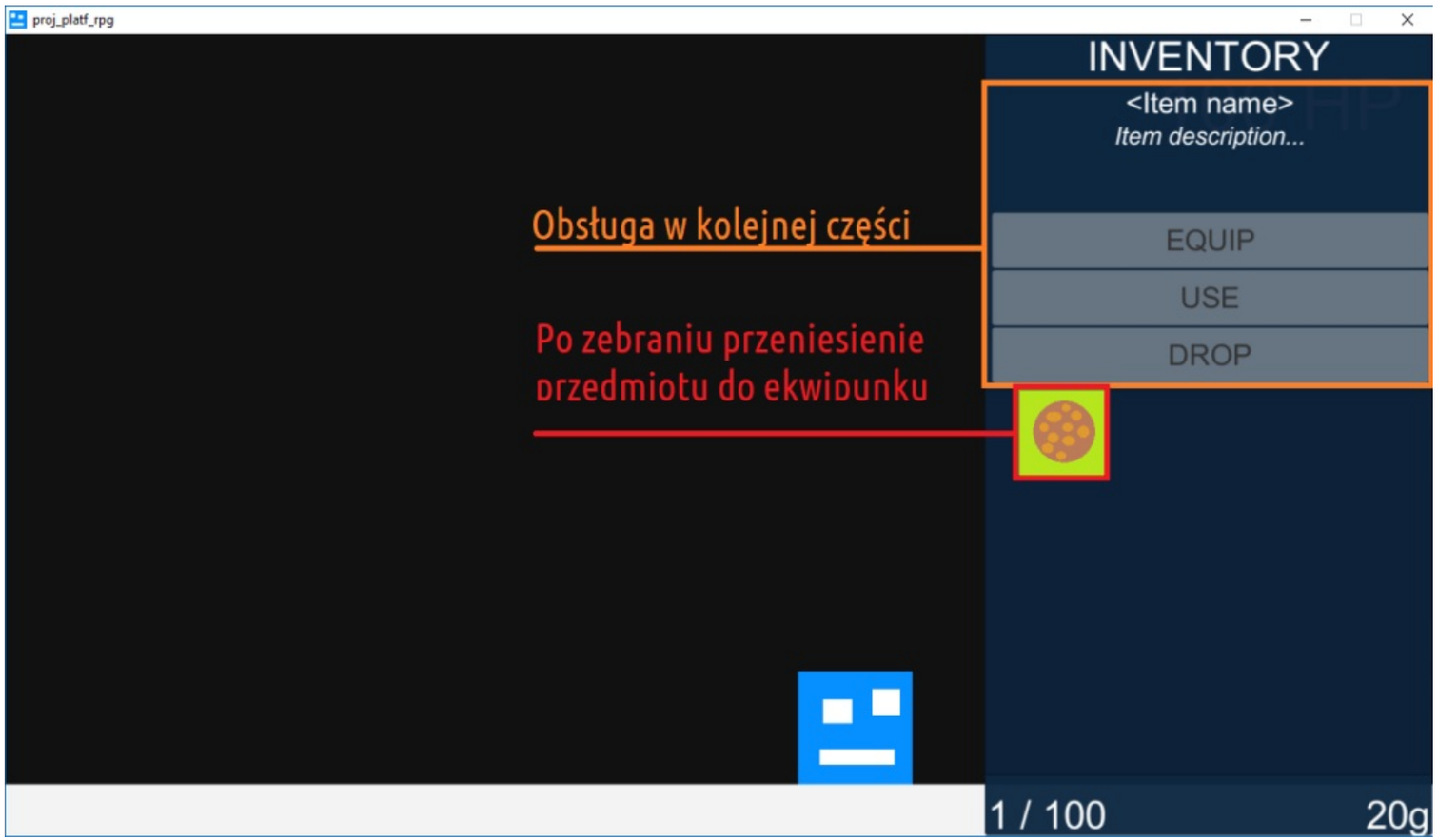
```
  public virtual void SetPhysicalOnScene(bool physical, Vector3 position)
  {
    // in general we are making just simple wrapper,
    // but in some cases we need to i.e. hide sprite renderer (weapons)
    m_itemObject.SetVisibleOnScene(physical, position);
  }
```

Jak widzimy na powyższym listingu – jest to wrapper, jednakże z całkiem dużym prawdopodobieństwem będziemy musieli w nim dodać dodatkową funkcjonalność, która będzie działa się w kontekście klasy *Item* (zniknie nam pusty wrapper).

Item.cs C#

```
274   protected virtual void Start()
275   {
276     foreach(ItemProperty prop in initialProperties)
277     {
278       enable_property(prop);
279     }
280
281     initialProperties = null;
282
283     m_itemObject = Instantiate(_physicalObjectPrefab)
284       .GetComponent<ItemObject>();
285
286     if (m_itemObject == null)
287       Debug.LogError("Empty Item Object", this);
288     else
289       m_itemObject.item = this;
290
291     SetPhysicalOnScene(_physicalOnInit, _physicalInitPosition);
292   }
```

Oprócz tego doszło nam stworzenie reprezentanta klasy *Item* przy tworzeniu obiektu, czyli jedyne co musimy dostarczyć to prefab i pozycję reprezentanta do klasy Item, nie musimy sami tej instancji tworzyć. Co ważne, to sam reprezentant może rozpocząć swój żywot w dwóch stanach: na scenie lub wewnątrz np. ekwipunku.



(https://i0.wp.com/szymonsiarkiewicz.pl/wp-content/uploads/2017/07/rgp-itemobject-aftercollect.png)

Po zebraniu przedmiotu

Bonus

Jako bonus chciałbym dodać kawałek kodu, który odpowiada za logikę ekwipowania przedmiotów, sam listing będzie omówiony szerzej w następnej części.

Item.cs C#

```
160   public bool SetEquipped(bool equip)
161   {
162     if(!HasProperty(ItemProperty.EQUIPABLE))
163     {
164       // Item cannot be equipped,
165       // so return error
166       return false;
167     }
168     else
169     {
170       bool isEquipped = HasProperty(ItemProperty.EQUIPPED);
171       if (isEquipped ^ equip)
172       {
173         if (equip)
174         {
175           // item is equipped and we weant to equip it...
176           enable_property(ItemProperty.EQUIPPED);
177         }
178         else
179         {
180           // item is equipped and we want to un-equip it
181           disable_property(ItemProperty.EQUIPPED);
182         }
183
184         return true;
185       }
186       else
187       {
188         // if both values are true or false...
189         return false;
190       }
191     }
192   }
```

Widzimy na nim, że pojawiła się nowa właściwość obiektu, która mówi czy dany przedmiot został wyekwipowany (EQUIPPED), ogólnie kod odpowiada za poprawną logikę nakładania / ściągania przez postać wyposażenia (a więc zapewnia, że założony przedmiot nie będzie założony 2 raz i w drugą stronę: nie można ściągnąć 2 razy tego samego wyposażenia).

Bug fixy

Zazwyczaj tego nie robię, ale tym razem chciałbym zauważyć, że względem ostatniej wersji mamy kilka dość istotnych fixów. Błędy (wynikające z nieuwagi) pozostawione w tej samej formie mogłyby powodować crashe (a wiem, że niektórzy równolegle do tego kursu robią własne gry, stąd ta notka).

Najbardziej wymowny jest diff, więc pozostawię je bez żadnych dodatkowych komentarzy:

1. [bugfix: value collision (https://github.com/sheadovas/proj_platf_rpg/compare/8adb363b8bb472fed4be224e33e9a3a740c8f522...6e82abb358423498c00543388fbc6be7af0e8ef5#diff-f1d9956c7dfd72548107506c88880b70L18)]
2. [bugfix: out of memory (https://github.com/sheadovas/proj_platf_rpg/compare/8adb363b8bb472fed4be224e33e9a3a740c8f522...6e82abb358423498c00543388fbc6be7af0e8ef5#diff-f1d9956c7dfd72548107506c88880b70L49)]

Podsumowanie

Dzisiaj zaimplementowaliśmy „fizyczną reprezentację” na przedmioty w naszej grze.

W kolejnej części (i miejmy nadzieję, że ostatniej dotyczącej ekwipunku) zajmiemy się funkcjami kontekstowymi przedmiotów (czyli tym co widać na screenie powyżej).

Tradycyjnie zachęcam do komentowania, dzieleniem się linkiem (do bloga, wpisu) ze znajomymi, pobrania dema i śledzenia tej (i innych serii) na blogu.

Code ON!