Piszemy RPGo-Platformówkę (12) – [Masz] i mój łuk! (sheadovas/poradniki/proj_platf_rpg/12-masz-i-moj-luk/)

Lip 18, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Dokańczamy integrację ekwipunku z UI.

Witam Was w ostatniej* części z UI. Dzisiaj zajmiemy się głównie interakcją przedmiotów z poziomu UI.

Na wstępie tradycyjnie zachęcam do ograniа [dema (https://github.com/sheadovas/proj_platf_rpg/releases/tag/1.9)] oraz własnej analizy dokonanych [zmian (https://github.com/sheadovas/proj_platf_rpg/compare/fae018fd388d68f633598af5cd713b96ed0dcdfb...7e1272868685deeb6a265efac1764b9d97c797d8)], którymi będziemy się dzisiaj zajmować.

(masz) I mój łuk!

Tym razem bez zbędnych ceregieli przejdźmy do praktyki, omówimy koncepcję nieco w biegu.

Equipment.cs C#

```
  public virtual void OnSelectItem(Item item)
  {
    update_selected(true, item);
  }

  public void EquipSelected()
  {
    // properly equip/unequip item
    m_selectedItem.SetEquipped(!m_selectedItem.HasProperty(Item.ItemProperty.EQUIPPED), GameMaster.gm.player);
  }

  public void UseSelected()
  {
    // HACK temporary solution
    // at this moment we have only one context for using item from menu
    // if more, then we have to choose in some way
    //
    // idea: lets add new item property "USABLE" and define new method "DefaultUseBehaviour"
    //   then each time we use item in USABLE ctx the DefaultUseBehaviour will be called
    m_selectedItem.Use(Item.ItemProperty.EATABLE, GameMaster.gm.player);
    update_weight(weight - m_selectedItem.baseWeight);

    if(m_selectedItem.quantity == 0)
    {
      // item is used & no items left in stack, so delete it from equipment
      DeleteItem(m_selectedItem.eid);
      update_selected(false);
    }
  }

  public void DropSelected()
  {
    DeleteItem(m_selectedItem.eid);
```

Nasza idea zakłada, że każdy kliknięty obiekt w ekwipunku wysyła wskaźnik na siebie, przez co sam obiekt zostaje „oznaczony” (co to oznacza dowiemy się za chwilę) – służy do tego metoda *OnSelectedItem*.

Same przyciski z ekwipunku połączone są z odpowiednio metodami: *EquipSelected*, *UseSelected*, *DropSelected*. Ich działanie ogranicza się do użycia odpowiednich metod ekwipunku lub przedmiotu. Metody wywoływane były omawiane szerzej w poprzednich wpisach.

Equipment.cs C#

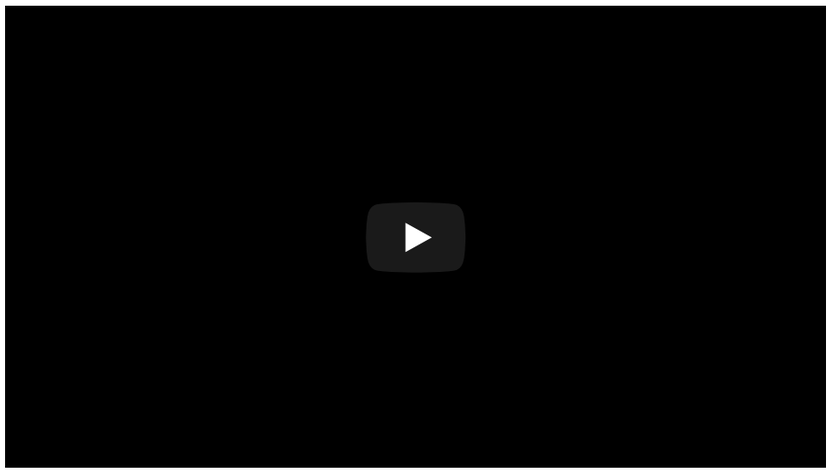
```
180   private void update_selected(bool selected, Item item = null)
181   {
182     foreach(Button button in m_actionButtons)
183     {
184       button.interactable = selected;
185     }
186
187     if(selected)
188     {
189       // fix buttons
190       if (!item.HasProperty(Item.ItemProperty.EQUIPABLE))
191         m_actionButtons[0].interactable = false;
192
193       if (!item.HasProperty(Item.ItemProperty.EATABLE) /* or any other "USABLE" property */)
194         m_actionButtons[1].interactable = false;
195     }
196
197     if(m_selectedItem != null)
198     {
199       // if we selected earlier something lets recolor it
200       m_selectedItem.GetComponent<Image>().color = new Color(1, 1, 1);
201     }
202
203     if (selected)
204     {
205       // recolor item to make item visually selected
206       item.GetComponent<Image>().color = new Color32(155, 155, 155, 255);
207
208       // update values
209       m_selectedTexts[0].text = item.itemName;
210       m_selectedTexts[1].text = item.itemDescription;
211     }
212     else
```

Wróćmy do oznaczania przedmiotów (czy raczej ich wybierania). Ta metoda jest uniwersalna także dla odłączania przedmiotów. Samo oznaczenie (zaznaczenie) przedmiotu zmienia obiekt wobec, którego będą działały przyciski.

Wyjaśnię powyższą mechanikę na przykładzie: jeżeli mamy w ekwipunku miecz i tarczę, a chcemy założyć miecz, to musimy najpierw kliknąć na ikonkę miecza, a następnie kliknąć przycisk EQUIP.

Powyższa metoda dba o odpowiednie aktywowanie przycisków (np. jeżeli przedmiot nie jest „używalny” to przycisk USE pozostanie nieaktywny) oraz o zaktualizowanie opisów dotyczących danego przedmiotu.

Działanie powyższego kodu wizualizuje film:



Kolejną relatywnie dużą zmianą było dodanie postaci wobec, której ma zostać użyty przedmiot. Przypomnę: do tej pory wywołanie metody *Use()* nie było połączane z żadną konkretną postacią – nie mieliśmy sposobu aby sprawdzić kto użył przedmiotu (mógł to być gracz, ale także dowolna inna postać).

ItemFood.cs

```
17  public void Eat(PlayableCharacter user)
18  {
19    Use(ItemProperty.EATABLE, user);
20  }
21
22  protected override void on_item_use(ItemProperty useContext, PlayableCharacter user)
23  {
24    switch(useContext)
25    {
26      case ItemProperty.EATABLE:
27        // only correct value
28        // TODO restore caller hp
29        eat(user);
30        break;
31
32      default:
33        // should be never called, because of guard in parent
34        Debug.LogError("Item cannot be used in following context!", this);
35        on_item_use_failure();
36        break;
37    }
38  }
```

Rozwiązaniem tego problemu było dodanie parametru *user*, który jest wskaźnikiem na postać używającą przedmiot. Parametr został ten dodany do wszystkich metod odpowiedzialnych za używanie przedmiotów.

ItemFood.cs

```
54  private void eat(PlayableCharacter user)
55  {
56    quantity--;
57    m_useLock = true;
58
59    user.stats.hp += GetRestorationHP();
60    GameMaster.gm.UpdateGUI();
61  }
```

Powyższy listing pokazuje, że w ten sposób łatwo jesteśmy modyfikować odpowiednie statystyki postaci, w przykładzie powyżej jest to poziom zdrowia.

ItemWeapon.cs C#

```
using UnityEngine;

public class ItemWeapon : Item
{
  public Weapon weapon;

  protected override void on_item_use(ItemProperty useContext, PlayableCharacter user)
  {
    switch(useContext)
    {
      case ItemProperty.EQUIPPED:
        // called when item is equipped / unequipped
        float modifier = 0;
        if(HasProperty(ItemProperty.EQUIPPED))
        {
          // item is now equipped, so we have to add stats
          modifier = weapon.dmg;
        }
        else
        {
          // item is unequipped, so delete effect from character
          modifier = -weapon.dmg;
        }

        // update stats
        user.stats.dmg += modifier;
        break;

      default:
        on_item_use_failure();
        break;
    }
  }
```

Ostatni listing pokazuje część łączącą broni z przedmiotami.

Tutaj wyjaśni się też gwiazdka z początku wpisu: integracja z *Weapon* nie jest pełna od strony Unity. Mianowicie żeby obecnie zaimplementowana broń była w pełni kompatybilna należałoby dopisać jeszcze trochę kodu i pobawić się z GUI. Ale skorzystam z przywileju, że traktuję Unity jedynie jako narzędzie do pokazywania pewnych pomysłów, to przynajmniej na razie daruję sobie pełną integrację (jeżeli ktoś ma ochotę, to zapraszam do puszczenia pull requesta) ;)

Podsumowanie

Udało nam się zakończyć temat ekwipunku i przedmiotów (yay!), dzisiaj sporo było rzeczy wymagających jedynie dobrego podejścia, aby rozwiązania problemów było dość łatwe do napisania ;)

W razie pytań, problemów, chęci podzielenia się opinią odsyłam do systemu komentarzy poniżej. Zachęcam jeszcze raz do ograna dema (link na początku wpisu) oraz do śledzenia tej serii oraz bloga (także przez social media).

Za tydzień zajmiemy się tzw. „questami” (zadaniami) i wykonamy sobie coś w stylu tradycyjnego „znajdź X przedmiotów, zabij Y potworów”.

Code ON!