Piszemy RPGo-Platformówkę (13) – Znajdź, zabij, posprzątaj (sheadovas/poradniki/proj_platf_rpg/13-znajdz-zabij-posprzataj/)

Lip 21, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Czyli zajmiemy się systemem zadań...

Hej, dzisiaj bez zbędnych wstępów zabierzmy się do dzieła! (oczywiście tradycyjnie zapraszam do samodzielnego zapoznania się ze [zmianami (https://github.com/sheadovas/proj_platf_rpg/commit/2e30c849d2a1949ee63cec7ea9b539fad0ffab0f)] oraz [demem (https://github.com/sheadovas/proj_platf_rpg/releases/tag/1.10)].

Fabularnie

Każda szanująca się gra RPG ma mniej lub bardziej rozwinięty system zadań. Samo zadanie zazwyczaj polega na wykonaniu jednej (lub większej ilości) prostych czynności, za które dostajemy nagrodę.

Przyjęło się, że zadania otrzymujemy od NPC (postaci niezależnych), ale wciąż się zdarza że takowe możemy po prostu zebrać z czegoś w rodzaju tablicy ogłoszeń.

Chyba nie muszę mówić, że dobrze zaprojektowane zadanie zapewnia pewne odczucie realności świata oraz pozwala na dogłębniejsze jego poznanie. Same zadania mogą składać się z wielu skomplikowanych kwestii dialogowych – świetnym przykładem takiego systemu jest seria gier Wiedźmin (kto nie grał niech się wstydzi ;P).

(https://i0.wp.com/szymonsiarkiewicz.pl/wp-content/uploads/2017/07/160127022344.jpg)
Tablica ogłoszeń – Wiedźmin 3

W tym wpisie nie chcę mówić o samym konstruowaniu zadań, my w tym projekcie zajmiemy się najbardziej oklepanym typem zadań, masowo wykorzystywanym w grach MMO. A więc...

Znajdź, zabij, posprzątaj...

Ten typ zadań to bardzo łatwo powtarzalna formuła, która uogólnia się do zapisu: „[Znajdź/Zabij/...] N [przedmiotów/potworów], otrzymasz za to X złota, Y doświadczenia, Z dodatkowych przedmiotów (np miecz)”. Każdy kto grał w MMO zna ten schemat dokładnie, podczas gry jedynie co się zmienia w takich zadaniach to ilość i nazwa interesującego nas obiektu.

A więc jak można podejść do zadań tego typu? Przede wszystkim należy zwrócić uwagę, że ogólny schemat jest powtarzalny, a jedynie co powinniśmy wyłapywać podczas gry to pojedyncze zdarzenia, które mogły być celem misji (np zbieranie przedmiotów).

Same zadania wyróżniają się jedynie typem obiektu i ich ilością, można by się było bawić w abstrakcję po klasie Zadanie dla każdego typu zadań (np zbieranie roślin, zabijanie zwierząt), tylko że nie ma takiej potrzeby. Demonstruje to kod poniżej:

Quest.cs C#

```cs
using System;
using UnityEngine;

[Serializable]
public class Quest
{
  public string questTitle = "Quest Title";
  public string questDescription = "i.e kill 10 monster to save the princess!";

  [Header("Objectives")]
  public int objectiveAmount = 1;

  // HACK Type of object would be a better idea,
  // but UnityEditor cannot handle with Type as configurable field from Editor :(
  public string objectiveTag;

  [Header("Rewards")]
  [SerializeField]
  protected int m_gold;
  [SerializeField]
  protected float m_exp;
  [SerializeField]
  protected Item[] m_items;

  protected bool m_isFinished = false;
  protected int m_collectedObjectives = 0;

  public virtual bool IsFinished(string updater, int amount)
  {
    // check if it is interesting type of item
    if (updater == objectiveTag)
    {
      // update quest details
```

Każde zadanie ma swoją nazwę oraz krótki opis. Oprócz tego przewidziana jest nagroda za jego ukończenie w postaci złota i/lub doświadczenia i/lub przedmiotów.

Za weryfikację ukończenia zadania służy metoda *IsFinished*, która jest wywoływana w momencie przyjścia zdarzenia. Jako parametry przyjmuje typ obiektu, który został np. zebrany oraz ilość tych obiektów (dopuszczamy w ten sposób do sytuacji gdy gracz zbierze kwiatków ze skrzyni jednocześnie). Oprócz tego umożliwia ona wcześniej ustalonej nagrody.

Zbieranie nagrody odbywa się wewnątrz metody *IsReward*, która zwraca nagrodę w momencie gdy zadanie zostało wykonane, w przeciwnym wypadku zwraca „nagrodę zerową”.

Questoard.cs C#

```
using UnityEngine;
using System.Collections;

public class QuestBoard : MonoBehaviour
{
  public Quest[] initialQuests;

  private Player m_player;
  private ArrayList m_activeQuests;
  private ArrayList m_questViews;

  [SerializeField]
  private Transform m_questBoardView;
  [SerializeField]
  private QuestView m_viewPrefab;

  public void NotifyNewEvent(string type, int amount=1)
  {
    for(int i=0; i<m_activeQuests.Count; ++i)
    {
      Quest quest = (Quest)m_activeQuests[i];
      if(quest.IsFinished(type, amount))
      {
        // acquire rewards
        int g;
        float exp;
        Item[] items;
        quest.GetReward(out g, out exp, out items);
        Debug.Log(string.Format("Finished Quest: {0}. Rewards: {1}g, {2}exp", quest.questTitle, g, exp));

        m_player.equipment.gold += g;
        // TODO add exp to player
        if (items != null)
```

„Tablica Zadań” posiada listę aktualnie wykonywanych zadań. Zajmuje się ona dodawaniem zadań do listy zadań oraz ich wyświetleniem na ekranie (*AddQuest*).

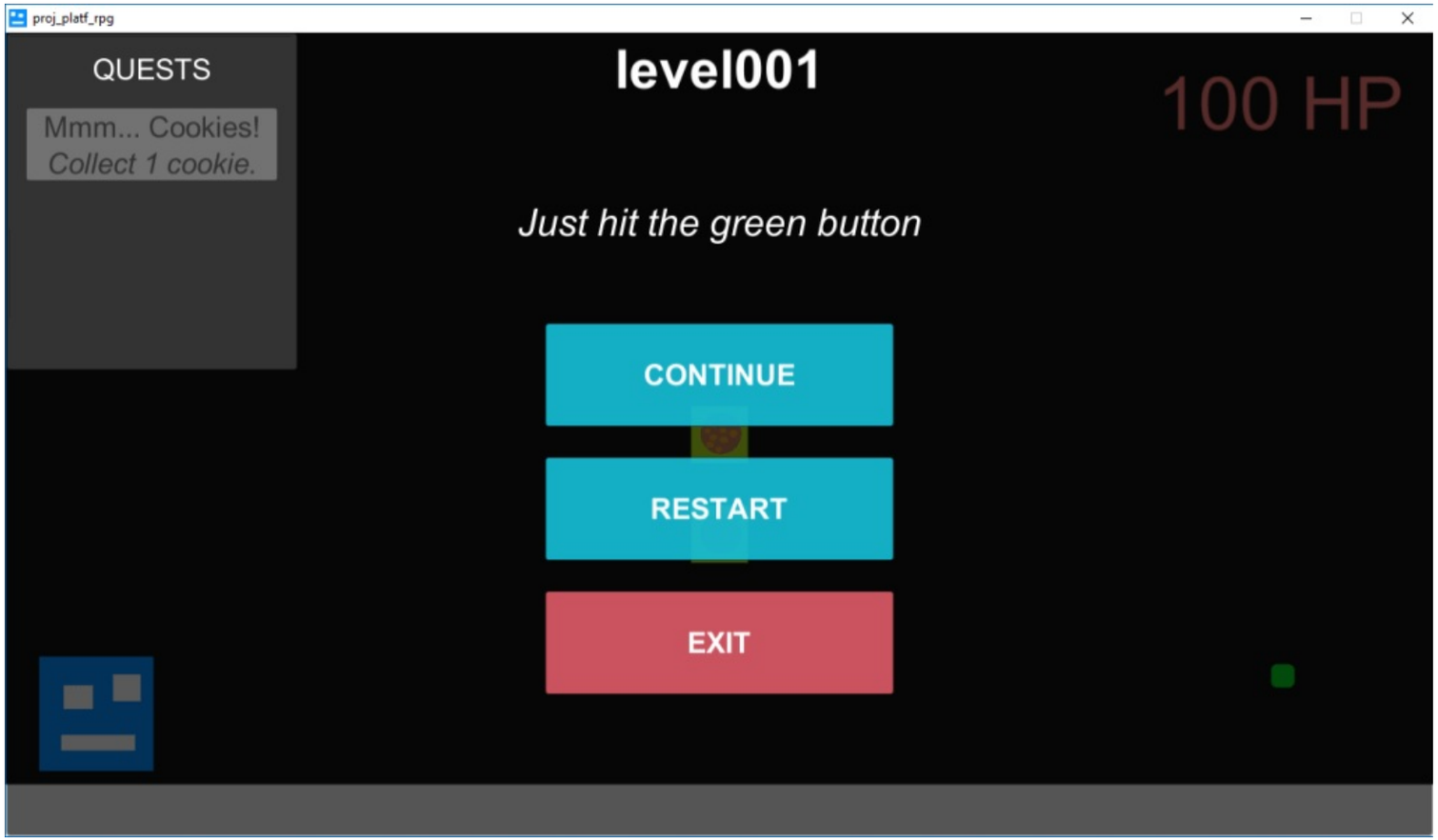
QuestBoard.cs

```
17  public void NotifyNewEvent(string type, int amount=1)
18  {
19    for(int i=0; i<m_activeQuests.Count; ++i)
20    {
21      Quest quest = (Quest)m_activeQuests[i];
22      if(quest.IsFinished(type, amount))
23      {
24        // acquire rewards
25        int g;
26        float exp;
27        Item[] items;
28        quest.GetReward(out g, out exp, out items);
29        Debug.Log(string.Format("Finished Quest: {0}. Rewards: {1}g, {2}exp", quest.questTitle, g, exp));
30
31        m_player.equipment.gold += g;
32        // TODO add exp to player
33        if (items != null)
34        {
35          foreach (Item item in items)
36            m_player.equipment.AddItem(item);
37        }
38
39        m_activeQuests.Remove(quest);
40      }
41
42      // <- here we can update progress (if we want)
43    }
```

Zdecydowanie ciekawszym elementem jest metoda *Notify*, która informuje tablicę o nowym zdarzeniu. Tutaj przeszukujemy listę aktualnie wykonywanych zadań i po kolei aktualizujemy zadania, poprzez ich sprawdzenie czy nie zostały wykonane – jeżeli tak, to pobieramy nagrodę oraz usuwamy zadanie z listy.

Warto też zauważyć, że gdybyśmy chcieli urozmaicić nasze zadania o bardziej skomplikowane funkcje, jak np. „przetrwaj X czasu”, to bez problemu możemy nadpisać metodę *Notify* i zmienić jej sposób działania (pozostając jednocześnie kompatybilnym z resztą kodu).

Uwaga! W powyższym listingu zaznaczyłem miejsce (linia 42) gdzie możemy dodać kod, który będzie służył do aktualizacji o postępie w zadaniu w GUI. Obecnie takowa informacja nie jest wyświetlana. Ten feature pozostawiam jako ćwiczenie do samodzielnego wykonania.



(https://i1.wp.com/szymonsiarkiewicz.pl/wp-content/uploads/2017/07/scrn_proj_platf_quests-1.png)

Teraz możecie powiedzieć: „dobra, dobra; wszystko fajnie, tylko skąd nadchodzi informacja o aktualizacji zadania?”. Pytanie jest z pozoru niewinnie proste, ale może spowodować mały mętlik w głowie.

Rozwiązanie jednak okazuje się dość proste: logujemy wszystkie zdarzenia, które możemy chcieć wykorzystać w zadaniach i je przekazujemy do tablicy zadań przez *NotifyNewEvent*. W swoim przypadku ograniczyłem się do rozpoznawania zbierania (dowolnych) przedmiotów:

PlayableCharacter.cs C#

```
275   protected void CollectItem(Item item)
276   {
277     GameMaster.gm.questBoard.NotifyNewEvent(item.tag, item.quantity);
278     item.SetPhysicalOnScene(false, item.transform.position);
279     equipment.AddItem(item);
280   }
```

I to tyle ;) W analogiczny sposób możemy dorobić wykrywanie zabijania potworów (to także zostawiam jako ćwiczenie).

W demie, które jest dostępne dla tej lekcji przygotowałem zadanie polegające na zebranie jednego ciasteczka (po czym otrzymuje się m.in. złoto), stworzyłem je z poziomu Unity poprzez dodanie do *initialQuests* zadania o nazwie „Mmm, Cookies!”, które polega na zebraniu 1 obiektu o type (tagu) „Cookie”, za co dostajemy złoto.

Podsumowanie

Jeżeli chodzi o prosty system zadań, to tyle co mogłem dla Was przygotować. Zdaję sobie sprawę, że można o nich opowiedzieć więcej, ale mówiąc szczerze to można zrobić o zadaniach w grach oddzielny poradnik (a to sobie na razie daruję). W każdej grze system zadań jest trochę inny, a ten który przedstawiłem na pewno stanowi dość popularną bazę pod zapotrzebowania większości gier, innymi słowy: eksperymentujcie (i pochwalcie się wynikami).

Oprócz tego (tradycyjnie) zapraszam do systemu komentarzy, gdzie możecie podzielić się swoją opinią, czy pytaniem. Nie zapomnijcie też śledzić bloga przez social-media (linki w panelu po prawej) oraz pochwalić się nim wśród znajomych (za co bardzo dziękuję).

W kolejnej części zajmiemy się drzewkiem umiejętności.

Code ON!