Piszemy RPGo-Platformówkę (9) – Witaj wędrowcze! (sheadovas/poradniki/proj_platf_rpg/9-witaj-wedrowcze/)

Cze 10, 2017 / proj_platf_rpg (sheadovas/category/poradniki/proj_platf_rpg/)

Wstęp do części z pisania gry RPG

Hej, dzisiaj zastanowimy się (ogólnie) jak powinna wyglądać nasza gra, tym razem w kontekście gier RPG.

Słowem wyjaśnienia...

Krótka notka odnośnie tego, że pojawiła się część dotycząca gry RPG, a nie ostatnia dotycząca platformówki:

W ostatniej części chciałem dorzucić ładniejsze animacje, lepszą grafikę, dźwięki, ... Ostatecznie uznałem, że nie mają one większego znaczenia, a same systemy prawdopodobnie mocno by się jeszcze pozmieniały, więc nie ma co tracić na to czasu i lepiej zabrać się za grę RPG, a te rzeczy dodamy nam sam koniec gry ;)

RPG

Gry RPG są jednym z trudniejszych gatunków gier w gamedevie: wymagają sporo pracy na wielu płaszczyznach – muszą mieć wiele dobrze zaprojektowanych mechanik (np walki), ale też tych „fabularnych”.

Sama gra powinna dawać wrażenie ‚prawdziwego’, żyjącego świata, rządzącego się logicznymi prawami, które łączą się w wiele mechanik dającymi swobodę graczowi: np aby zdobyć miecz (**ekwipunek**):

1. Możemy go kupić (**handel**), z kolei do tego potrzebujemy pieniędzy; te z kolei możemy zdobyć wykonując zadania (**misje**).
2. Inną ścieżką do zdobycia miecza jest jego wykucie (**crafting**), tutaj z kolei wymagany jest pewien poziom umiejętności wykuwania broni (**drzewko umiejętności**).
3. Oczywiście nic nie stoi na przeszkodzie aby go ukraść (**system dobra i zła** / **morale**).

Jak widzimy do tak prostej czynności możemy podejść na wiele sposobów, taka swoboda zapewnia graczowi sporo frajdy, ale nam – programistom – dodaje całkiem sporo systemów do napisania, po to tylko aby mógł zdobyć miecz w dowolny sposób ;)

Witaj wędrowcze!

Mam nadzieję, że powyższym przykładem chociaż trochę poruszyłem Twoją wyobraźnię, odnośnie tego jak skomplikowane mechaniki mogą kryć się za prostymi czynnościami. W ciągu kolejnych lekcji chciałbym się nimi zająć, ale tutaj ważna uwaga: nie wszystkie poruszymy dogłębnie – po prostu stworzymy jej uproszczoną wersję i przejdziemy dalej (powód? zamieniłoby się to pisanie w pełnoprawnej gry RPG, a pisanie pełnoprawnej gry RPG w pojedynkę nie jest za dobrym pomysłem).

Wszystkie systemy, które chciałbym omówić:

- ekwipunek – plecak gracza,
- handel – pieniądze, sklepy, handlarze,
- zadania (misje),
- umiejętności (rozwój postaci),
- morale,
- NPC – rozmowy, opcje dialogowe,
- (opcjonalnie) crafting.

Jeżeli macie jeszcze jakieś pomysły to zapraszam Was do pisania ich poniżej, być może nimi także się zajmę.

> Uwaga: Nie zdarzyło mi się jeszcze „popełnić” żadnej gry RPG, część z tych systemów już zdarzyło mi się pisać (np umiejętności), także należy mieć to na uwadze, że prezentowane przeze mnie rozwiązania są autorskie i nie muszę się w 100% pokrywać z tym jak to faktycznie wygląda w gamedev’ie.
>
> Ta część „poradnika”, zamienia się we wspólną podróż gdzie razem odkrywamy „nowe lądy”, dlatego tym bardziej zachęcam do współpracy nad kodem, dzieleniem się pomysłami, spostrzeżeniami ;)

Poniżej przedstawiam ogólny pogląd na elementy do zaimplementowania:

```
Szkic kodu
```
C#

```
/*
Equipment
  used by:
  - Character (Player, NPC)
  - Shop
  ...
*/
class Equipment {
  int capacity;      // max of "kg" or "item count" in bagpack
  int free_capacity; // currently free capacity (= capacity - used)
  Item[] items;      // collection of items in backpack
}

/* Item (abstract) */
// abstract base for all other items
// i.e. weapons, armor, quest items, ...
class Item {
  string name;       // name, short description
  string type;       // type of item (weapon, food)

  int quality;       // quality of item (rusty, normal, epic), i.e "Epic Sword"
  int base_cost;     // base cost of item, can be manipulated by "durability"
  int durabilty;     // when 0 -> item becomes broken, in weapon: multiplier for dmg

  int cu;            // capacity units: mass or size (depends on used model in EQ)
  int amount;        // amount of items int stack (only if item is stackable)
  int stack_size;    // max count of items in stack
};

/* Shop */
class Shop {
  Item[] items;      // items in shop stock
  float multiplier;  // cost multiplier, depends on ie player morale
```

Podsumowanie

To koniec na tą część, liczę że tematy które chciałbym poruszyć wydają się Wam ciekawe. Zachęcam do przestudiowania podrzuconego przeze mnie szkicu kodu i podzielenia się swoją opinią, własnymi propozycjami.

Tradycyjnie zapraszam do sekcji komentarzy, śledzenia bloga przez social-media (panel po prawej) i dzielenia się tym wpisem ze swoimi znajomymi.

W kolejnej części zabierzemy się za ekwipunek, do przeczytania w kolejnym wpisie!

Code ON!