Temat pracy: Stworzenie gry RPG z użyciem grafiki 2D.

Paweł Kamiński

7 czerwca 2015

1 Zarys oczekiwanego efektu końcowego

Celem jest stworzenie gry 2D zawierającej mechanikę gry inspirowaną grami z serii Final Fantasy.

2 Planowane moduły gry

- mapa świata
- mapy lokalne
- pole bitwy
- ekran menu głównego
- ekran rozwoju postaci
- ekran ekwipunku
- ekran ustawiania opcji
- ekran wczytywania gry
- ekran zapisywania gry

Przełączanie pomiędzy poszczególnymi modułami będzie możliwe dzięki użyciu State Machine.

3 Założenia ogólne

Gracz może poruszać się pomiędzy kilkoma lokacjami, wybierając lokację, do której chce się udać, z mapy świata. Po wybraniu danej lokacji gracz może poruszać się po mapie lokalnej. Na mapie lokalnej może dojść do następujących wydarzeń:

- Gracz opuszcza mapę lokalną korzystając ze zdefiniowanego wyjścia. W takim wypadku gracz jest przenoszony na mapę świata.
- Gracz napotyka przeciwnika. W takim wypadku gracz jest przenoszony na pole bitwy.
- Gracz napotyka NPC (inną postać). W takim przypadku:
  - jeżeli jest to postać znacząca, przeprowadza dłuższy dialog z daną postacią, może dostać zadanie lub ukończyć zadanie
  - jeżeli jest to postać nieznacząca, może przeczytać krótkie zdanie wypowiadane przez daną postać, bez wpływu na przebieg fabuły
- Gracz znajduje przedmiot. W takim przypadku przedmiot dodawany jest do ekwipunku.

4 Założenia odnośnie zadań w grze

Gra nie będzie zawierała zadań pobocznych. Zadania będą mogły być wykonywane wyłącznie w ustalonej kolejności.

5 Założenia odnośnie systemu walki w grze

Planowany system walki to turowy system walki z elementami czasu rzeczywistego. W czasie jednej walki postać sterowana przez gracza mierzy się z pewną liczbą przeciwników (od 1 do 3). Każda postać posiada punkty życia (healthPoints). Niektóre postaci mogą posiadać dodatkowo punkty magiczne (magicPoints). Gracz i przeciwnicy mogą wykonywać działania tylko wtedy, gdy mają swoją turę.

W przeciwieństwie do klasycznego turowego systemu walki, tury nie muszą występować na zmianę. Każda z postaci posiada licznik odliczający czas wstecz do możliwości wykonania następnego ruchu (remainingWaitingTime). Informacja o czasie pozostałym do wykonania ataku przez przeciwnika jest jawna dla gracza. Gdy któryś licznik osiągnie 0, dana postać ma prawo wykonać akcję. Niektórzy z napotkanych przeciwników mogą być szybsi lub wolniejsi od gracza w swoich działaniach, tzn. każda postać posiada swój własny współczynnik czasu oczekiwania (waitingTime).

Celem akcji wykonywanej w czasie swojej tury będzie zazwyczaj wykonanie ataku odbierającego punkty życia przeciwnikowi (damage). W zależności od etapu gry, mogą być możliwe różne inne rozwiązania, takie jak przyspieszanie działań własnej postaci, spowalnianie działań przeciwnika, rzucanie czarów defensywnych redukujących przyszłe obrażenia itp. Walka będzie polegać na analizowaniu obecnej sytuacji i wybieraniu jak najskuteczniejszej akcji spośród listy dostępnych akcji.

Każda z akcji posiada pewien czas potrzebny na jej przygotowanie. Jeżeli postać zostanie zaatakowana w trakcie przygotowywania swojego ataku, jej tura

przepada. Zazwyczaj akcje wymagające więcej czasu są potężniejsze, jednak ich wybieranie wiąże się z większym ryzykiem, gdyż przeciwnik może w tym czasie wykonać atak. Z punktu widzenia gracza istotne jest analizowanie współczynnika remainingWaitingTime dla każdego z przeciwników i dobieranie dłuższych ataków wtedy, gdy jest jeszcze dużo czasu do tury przeciwnika.

Każda postać posiada współczynnik obrony (defence). Jest on wyrażony w skali od 0 do 100 i określa, ile procent obrażeń zadawanych przez przeciwnika jest unikanych. Ilość punktów odbieranych przez atak jest wyrażona wzorem: effectiveDamage = damage * (100 – defence)

Jeżeli effectiveDamage jest większe niż ilość punktów życia posiadana przez postać, która została zaatakowana, odbierana jest liczba punktów równa ilości punktów życia zaatakowanej postaci, tak aby liczba punktów życia nigdy nie była ujemna.

Po każdej turze następuje sprawdzenie, czy liczba punktów życia gracza lub wszystkich przeciwników wynosi 0. Liczba punktów równa 0 oznacza śmierć postaci.

Jeżeli liczba punktów życia wszystkich przeciwników wyniesie 0, walka jest zakończona sukcesem. Gracz jest przenoszony z powrotem do mapy lokalnej. Jeżeli liczba punktów życia gracza wyniesie 0, walka jest zakończona porażką. Gracz otrzymuje propozycję podjęcia ponownej próby.

6 Dźwięk

Gra będzie posiadała ścieżkę dźwiękową odtwarzaną w czasie eksploracji, walk oraz na ekranach menu. W tym celu planuję wykorzystać utwory udostępnione publicznie na wolnych licencjach. Rozważam również wykorzystanie dźwięków związanych z wykonywaniem konkretnych działań w czasie walki lub przy znalezieniu przedmiotu. Wszelkie dialogi w grze będą dostępne wyłącznie w formie tekstowej.

7 Rozwój postaci

Wraz z postępami w odkrywaniu fabuły gracz dostanie do dyspozycji nowe umiejętności. Gracz może zobaczyć odblokowane umiejętności na ekranie rozwoju postaci. Rozwój postaci jest liniowy i ściśle powiązany z fabułą. Gracz nie ma możliwości wybierania umiejętności do rozwoju, a ekran rozwoju postaci służy jedynie do sprawdzenia, jakie umiejętności są już odblokowane.

8 Platforma docelowa

Gra jest skierowana na komputery osobiste z systemami z rodziny Linux oraz Windows.

9 Technologia używana do wykonania projektu

Technologia, którą chciałbym wykorzystać to język Lua (http://www.lua.org/) oraz framework LÖVE (https://love2d.org/) do języka Lua. Jeśli chodzi o tworzenie map, chciałbym wykorzystać edytor map Tiled (http://www.mapeditor.org/).

Dlaczego wybrałem taką technologię? Framework LÖVE jest na wolnej licencji, umożliwiającej wykorzystywanie projektu w dowolnym celu. Kod pisany przy użyciu LÖVE działa na Linuksie, Mac OS X oraz Windows. W porównaniu do np. C++ nie trzeba się zajmować wieloma niskopoziomowymi operacjami. Posiada rozbudowaną i dobrze udokumentowaną bibliotekę HUMP (LÖVE Helper Utilities for Massive Progression), która umożliwia wykonywanie przejść pomiędzy poszczególnymi modułami gry (State Machine). Wykonanie typowych czynności, takich jak np. odtworzenie pliku muzycznego sprowadza się do kilku linijek kodu.

10 Opisanie zawartości plików

10.1 battle.lua

Plik battle.lua zawiera funkcje:

battleState:update(dt) - funkcja zajmująca się głównie odliczaniem czasu

resetCounter(o1, o2, o3) - funkcja przywracająca domyślne wartości czasu oczekiwania na rozpoczęcie tury

colourIfNeeded(expectedValue) - funkcja zmieniająca kolor czcionki w zależności od tego, czy tekst jest wskazywany przez strzałkę

10.2 battleBackend.lua

Plik battleBackend.lua zawiera funkcje:

fight(o1, o2, o3) - funkcja obsługująca liczniki do rozpoczęcia tury gracza oraz przeciwników

setArrow() - funkcja zapobiegająca wskazywaniu strzałką na martwych przeciwników

isDead(o1) - funkcja sprawdzająca, czy ktoś jest martwy

setInitialValues(tmpValues, initialValues) - funkcja przywracająca domyślne wartości atrybutów postaci

10.3 battleBackendMakeAction.lua

Plik battleBackendMakeAction.lua zawiera funkcję:

makeAction(selectedAction) - funkcja wykonująca akcję o indeksie zawartym w argumencie oraz sprawdzająca, czy walka powinna już zakończyć się zwycięstwem

10.4 battleBackendTurns.lua

Plik battleBackendTurns.lua zawiera funkcje:
playerTurn() - funkcja obsługująca rozpoczęcie się tury gracza
enemyTurn(o1) - funkcja obsługująca całość przebiegu tury przeciwnika

10.5 battleBackendWinning.lua

Plik battleBackendWinning.lua zawiera funkcję:
winning() - funkcja obsługująca zakończenie walki zwycięstwem gracza

10.6 battleControls.lua

Plik battleControls.lua zawiera funkcję:
battleState:keypressed(key) - funkcja obsługująca sterowanie za pomocą klawiatury w stanie walki

10.7 battleControlsMoveArrow.lua

Plik battleControlsMoveArrow.lua zawiera funkcje:
moveArrowDown() - funkcja ustawiająca odpowiednio współrzędne strzałki wskazującej na przeciwnika w przypadku naciśnięcia klawisza strzałki w dół
moveArrowUp() - funkcja ustawiająca odpowiednio współrzędne strzałki wskazującej na przeciwnika w przypadku naciśnięcia klawisza strzałki w górę

10.8 battleDraw.lua

Plik battleDraw.lua zawiera funkcję:
battleState:draw() - funkcja odpowiadająca za wyświetlanie klatki obrazu

10.9 battleLoading.lua

Plik battleLoading.lua zawiera funkcję:
battleState:enter() - funkcja odpowiadająca za wczytanie plików graficznych i muzyki

10.10 enemy1.lua

Plik enemy1.lua zawiera funkcje:
E_attack() - funkcja odpowiadająca za wykonanie zwykłego ataku przez Dorvera
E_magicAttack(attackingEnemy) - funkcja odpowiadająca za wykonanie magicznego ataku przez Dorvera
E_heal(castingEnemy) - funkcja odpowiadająca za uleczanie się Dorvera
E_defend(castingEnemy) - funkcja odpowiadająca za bronienie się Dorvera
Enemy1AI() - niezaimplementowana, docelowo ma zawierać algorytm działań w czasie tury Dorvera, sztuczną inteligencję przeciwnika

10.11 main.lua

Plik main.lua to główny plik gry. Zawiera funkcje:

love.load() - funkcja wczytująca ustawienia, takie jak czcionka, domyślna pozycja gracza itp.

love.keypressed(key) - funkcja zawierająca obsługę poleceń klawiaturowych działających globalnie w całej grze

love.draw() - funkcja wywoływana przed wyświetleniem każdej klatki, dostosowuje wyświetlany obraz do rozdzielczości ekranu

10.12 map.lua

Plik map.lua zawiera funkcje:

mapState:update(dt) - funkcja zajmująca się uaktualnianiem mapy i w razie potrzeby przełączeniem w stan walki

mapState:draw() - funkcja wyświetlająca mapę

mapState:keypressed(key) - funkcja obsługująca sterowanie w stanie chodzenia po mapie

10.13 mapEnterState.lua

Plik mapEnterState.lua zawiera funkcję:

mapState:enter() - funkcja odpowiadająca za wczytanie plików graficznych do wyświetlenia na mapie

10.14 player.lua

Plik player.lua zawiera funkcje:

attack(o1) - funkcja odpowiadająca za wykonanie przez gracza zwykłego ataku

magicAttack(o1) - funkcja odpowiadająca za wykonanie przez gracza magicznego ataku

lockedAction(o1) - ta funkcja nic nie robi. Jest obecna w celach testowych.

heal() - funkcja odpowiadająca za rzucanie czaru uleczającego

defend() - funkcja odpowiadająca za akcję obronną

listOfAllActions() - funkcja odpowiadająca za stworzenie tabeli zawierającej wszystkie dostępne akcje (tzn. z pominięciem tych, które są zablokowane)

10.15 playerActionFlags.lua

Plik playerActionFlags.lua nie zawiera żadnych funkcji. Zawarta jest tu tabela zawierająca nazwy akcji, informacje o odblokowaniu akcji, id, opis, liczba potrzebnych punktów magicznych do wykonania oraz czy można obecnie wykonać akcję.