Dwarven Treasure

by Jakub Froń

Dokumentacja programisty

1. **Ogólne informacje**
   Projekt pisany był bez wykorzystania żadnego IDE, czysto Sublime Text 3. Kompilowany i uruchamiany za pomocą terminala w podsystemie Ubuntu postawionym pod Windows 10.
2. **Zestaw klas wraz z ich opisem**

- **Main:**
  Game
  Odpowiada tylko za stworzenie GameFrame.

  GameFrame
  Rozszerza JFrame. Odpowiada za stworzenie nowego GamePanel i dodanie go do widoku.

  GamePanel
  Rozszerza JPanel implementuje Runnable i KeyListener
  Przechowuje wymiary okna gry.
  Odpowiada za stworzenie pętli gry, kontrolowanie głównego wątku, nasłuchiwanie akcji użytkownika (wciśnięcie któregoś z klawiszy ruchu/spacji).

  void addNotify() - uruchamia się automatycznie przy inicjalizacji, dodaje nowy keyListener oraz tworzy i uruchamia główny wątek.

  void run() - uruchamia funkcję init() oraz kontroluje pętlę gry, czyli sprawdza czas, który zajęło stworzenie klatki gry i synchronizuje to z framerate'em.

  void init() - tworzy BufferedImage, czyli obraz, oraz przypisuje mu Graphics2D, czyli artystę. Tworzy również GSManager, czyli obiekt, który zarządza stanami gry. Zostanie to dalej wytłumaczone w części Management.

  Pozostałe metody odpowiadają za wywoływanie aktualizacji obrazu, obsługi inputu i rysowania w obiektach niżej.

- **Management:**
  GSManager
  Zarządca stanów gry, które dalej zostaną wytłumaczone w części States. Odpowiada za zmiany stanów wynikające z różnych faz programu. Podczas konstrukcji automatycznie uruchamia stan Intro, po jego zakończeniu przełącza na stan Menu, z Menu można dalej przejść do stanu faktycznej rozgrywki itd..
  Przechowuje tablicę typu GameState w której przechowuje wszystkie stany stworzone przez programistę. Dodawanie nowych stanów wymaga określenia ich w tej klasie i dodania ich obsługi.

  Zawiera metody zmiany stanu, oraz aktualizacji i rysowania obecnego stanu.

  KeysHandler
  Zarządca interakcji przez klawiature.
  Nadzoruje wewnętrzną tablicę wciśniętych przycisków i na tej podstawie udostępnia interfejs sprawdzania, czy konkretny lub czy jakikolwiek klawisz jest wciśnięty.

  Graphics
  Zadaniem klasy graphics jest przygotowanie grafik, które będą wykorzystywane przez inne klasy.

  BufferedImage[][] load(..) Przyjmuje ścieżkę do pliku, oraz dwie wielkości, które mówią mu, na jakie elementy ma podzielić podany plik.

  Przykład:
  … font = load('font.png', 8, 8) zwróci tablicę typu BufferedImage, w której znajduje się cały plik font.png podzielony na elementy 8x8.

  void drawString(...) Przetwarza podany do wypisania string i na jego podstawie wybiera z tablicy font[][] potrzebne 'wycinki' – znaki i wypisuje je na ekran zaczynając od podanych współrzędnych x, y i przesuwając się wzdłuż osi x o 8 pikseli (dokładnie tyle ile ma jeden znak).

- **Map:**
  Map
  Zadaniem tej klasy jest załadowanie grafik „płytek” do tablicy, załadowanie danych odnośnie mapy z pliku txt oraz zarządzanie tym w jaki sposób jest rysowana i aktualizowana.

W celu oszczędzania pamięci wykorzystywana jest w jednym momencie prawie najmniejsza wymagana ilość informacji o mapie, 8x8 płytek + offset wielkości 2 na wszelki wypadek. Obiekt tej klasy przechowuje zmienne odpowiedzialne za offset od którego ma zacząć szukanie informacji w swoich strukturach mapy i je wyrysowywać. Więc zamiast za każdym razem rysować pełną mapę, klasa wie gdzie jest teraz gracz, lub gdzie gracz chce przejść i rysuje tylko ten segment na którym stoi.

Tile
Przechowuje tylko dwie informacje: obrazek tej płytki i jej typ (zablokowana/odblokowana – czy gracz może na nią wejść).

- **States:**
  GameState
  Abstrakcyjna klasa, którą dalej tylko rozszerzają klasy stanów gry. Wymaga od klasy implementacji obsługi inicjalizacji, aktualizacji klatki, rysowania i przetwarzania inputu. W inicjalizatorze przypisuje sobie też nadrzędny GSManager.

  IntroState
  Stan, który uruchamia się jako pierwszy. Całym jego zadaniem jest wyświetlenie logo z efektem fade-in i fade-out. Można go pominąć wciśnięciem któregokolwiek z wymienionych wcześniej klawiszy obsługi gry. Po zakończeniu odnosi się do GSManager'a i żąda przełączenia na stan MENU.

  MenuState
  Inicjalizacja tej klasy ładuje grafiki wykorzystywane w tym obiekcie. Klasa odpowiada za narysowanie tła, dwóch opcji wyboru oraz obsługę zdarzeń ze strony użytkownika (zmiana podświetlanej opcji lub wybranie opcji). Wybór START powoduje przełączenie na stan PlayState, wybór EXIT kończy rozgrywkę.

  PlayState
  Odpowiada za inicjalizacje wielu obiektów, w tym Map, Player, Item, GoldNugget.
  W swoim inicjalizatorze wywołuje metody załadowania mapy, stworzenia gracza i umieszczenia go na mapie i ogólnie za spawn przedmiotów i przypisanie początkowego sektora do rysowania.
  Metody update i draw głównie wywołują te same metody w obiektach niżej. W klasie update sprawdzane jest położenie gracza i dane te przekazywane są dalej do obiektu Map. Sprawdzane jest to, czy gracz

wszedł na pole z interaktywnym przedmiotem i na tej podstawie uruchamiane są metody gracza, np. collectedGold().
Metoda processInput bada, które klawisze są wciśnięte i przekazuje te informacje do klasy gracza. Po spełnieniu kryteriów rozgrywki następuje przejście do stanu GameOverState.

GameOverState
Tu jedyne co się dzieje, to wypisanie krótkiego podziękowania za rozgrywkę. Naciśnięcie enter powoduje zamknięcie gry.

- **HUD:**
  Hud
  Hud to pasek w dole ekranu, który pojawia się tylko w stanie PlayState, informuje on gracza o postępie, czyli o ilości zebranych sztab złota oraz czy gracz posiada przedmiot kilof. Podniesienie sztaby złota lub zdobycie kilofa powoduje zaktualizowanie danych wyświetlanych przez Hud.
- **Entity:**
  Entity
  Abstrakcyjna klasa wyznaczająca wymogi, które spełnić mają obiekty ją rozszerzające, takie jak GoldNugget czy Player. Wyznacz potrzebę implementacji intersekcji obiektów (gracz zbiera złoto) oraz getterów i setterów odnośnie parametrów np. pozycji na mapie.

  Player
  Obiekt klasy Player przechowuję m.in. zestaw swoich sprite'ów, ilość zdobytego złota, ilość całkowitego złota, wartość logiczną prawda/fałsz określającą posiadanie kilofa.
  setAction() **-** gdy gracz stoi obrócony do skruszonej ściany, posiada kilof i naciska spację, zamieniane jest pole, które zawierało skruszoną ścianę na zwykłe pole do przejścia.
  update() - sprawdza wartości boolowskie takich zmiennych jak down/up/left/right, które odpowiadają za obecny kierunek gracza i zestawiają to z obecnie odtwarzaną animacją, by określić czy trzeba zmienić obecną animację na inną. Odnosi się też do rozszerzanej przez siebie klasy Entity, która określa miejsce docelowe postaci i na tej podstawie zmienia jej położenie.

  GoldNugget/Item/Graal
  Rozszerza klase Entity, dodaje tylko swoje grafiki do zestawiu do animacji i wszystkie żądania aktualizacji i rysowania przekazuje dalej do rozszerzanej przez siebie klasy. Item dodatkowo przekazuje informacje o tym, czy został zebrany, by ta informacja mogła zostać wykorzystywana

do pominięcia go w rysowaniu następnej klatki na mapie.

Animation
Ta klasa nie rozszerza klasy Entity.
Zawiera w sobie tablice klatek do rysowania, oraz algorytm przełączania ich na podstawie licznika animacji i opóźnienia.

3. Podsumowanie

Projekt pisany był z myślą o jego przyszłej rozbudowie, taki też mam zamiar. W niektórych miejscach można zauważyć kod, który teoretycznie nie jest potrzebny, lub rozwiązania, które zdają się utrudniać cały projekt. Jest to w 90% celowe zamierzenie.