Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

AGH University of Science
and Technology

Gra RPG na Androida

Jakóbczak Dawid, Stoch Mikołaj

21 maja 2019

Skąd pomysł na temat projektu?

- Android - dla nas całkowite nowe środowisko, które warto poznać
- Tworzenie gry daje dużą swobodę

Cele, czyli to co chcieliśmy osiągnąć

- Zdobycie podstawowej wiedzy na temat pisania aplikacji na Androida
- Zapoznanie się z technikami używanymi podczas pisania prostych gier
- Rozwinięcie umiejętności programowania w języku Java

Android Studio vs Intellij

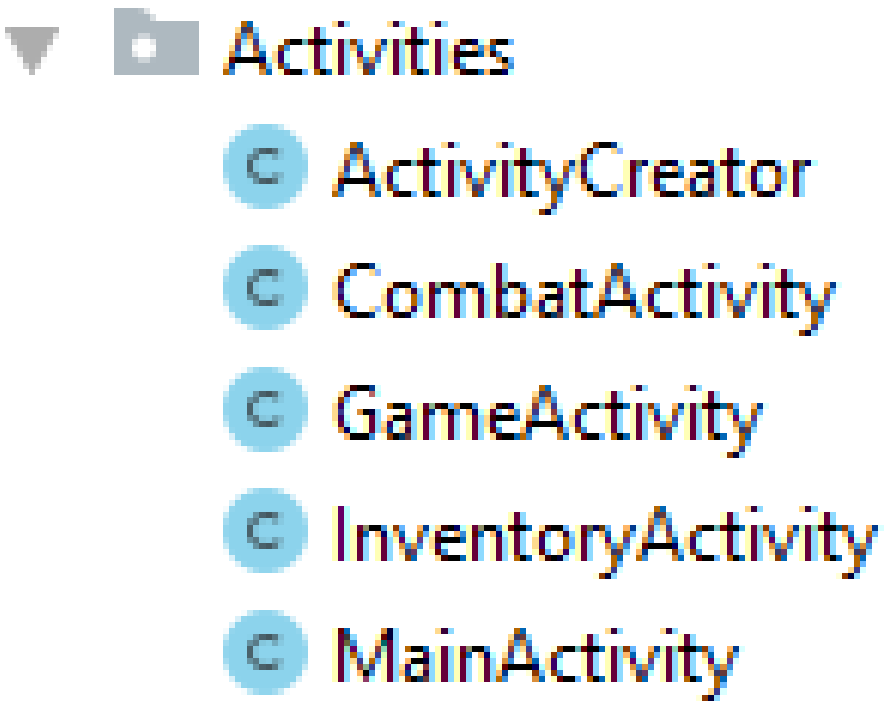
Android Studio

- JetBrains and Google
- Używałeś Intellij IDEA? Jeśli tak to z Android Studio (prawie) nie będziesz miał problemów.
- Darmowy, rozwijany od 2013r.
- Zamiast terminala, emulator Androida.
- Log.d

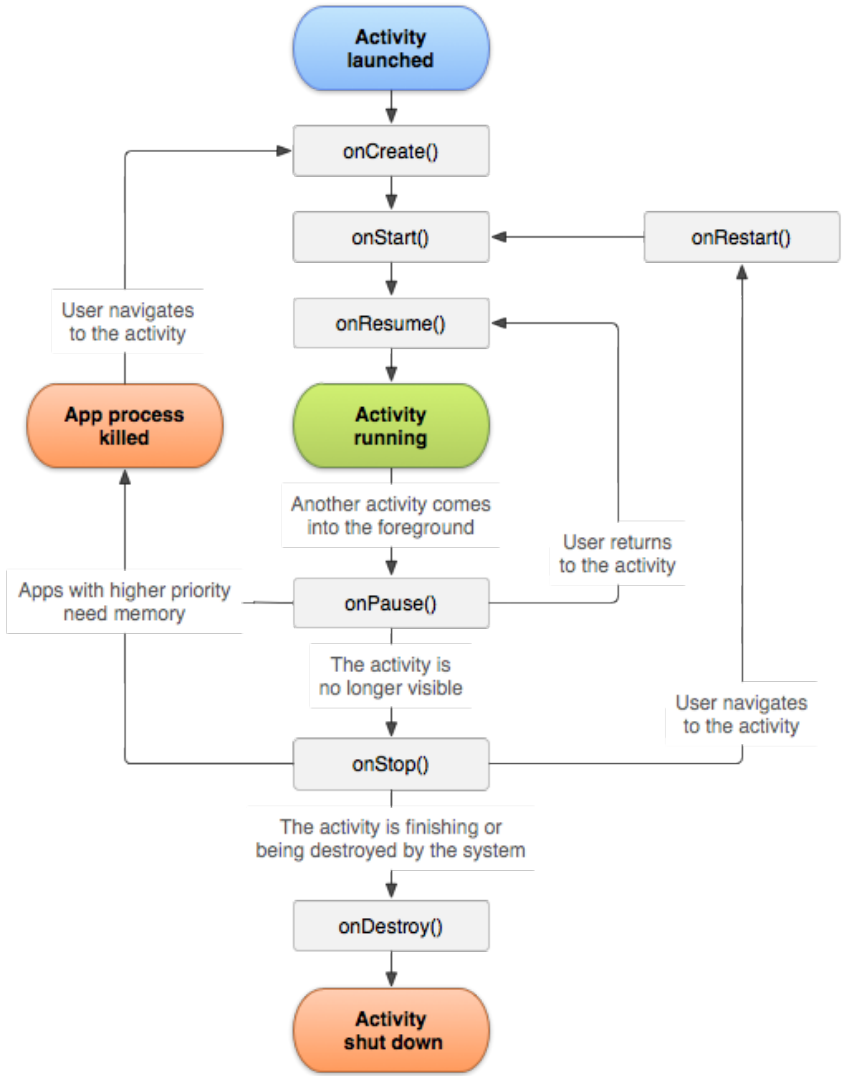
Ekran główny

Activities

- Na nich oparta jest cała aplikacja w systemie Android
- "Przeskakujemy" z różnych Activities
- onStart(), onCreate(), onStop() itd..



Cykl życia Activity



Activities

- MainActivity
- Powiązanie activity z plikiem .xml - tak zwany conent
- "Hello World" -> activitymain.xml + MainActivity.java
- setContentView()
- Definiujemy atrybuty wyświetlania naszej aplikacji

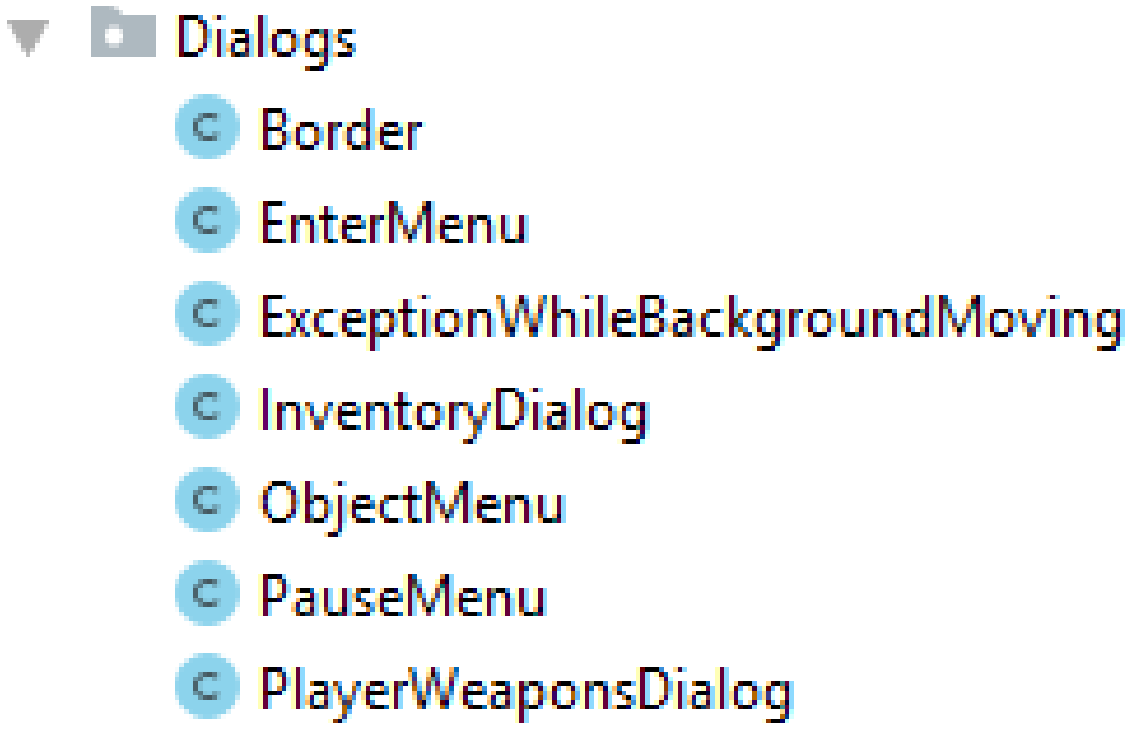
Atrybuty Activity

```
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
this.setRequestedOrientation( ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

GameView

- Dziedziczy z klasy SurfaceView - specjalna klasa służąca do rysowania/renderowania obiektów
- W naszym projekcie główna klasa
- W niej nasłuchujemy na "tapnięcie" ekranu
- Metody update(), onTouchEvent(), draw()

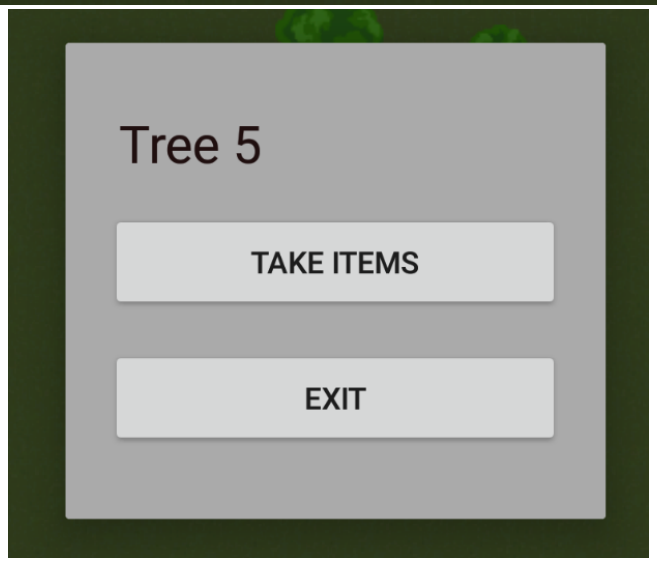
Dialogi


Dialogs
Border
EnterMenu
ExceptionWhileBackgroundMoving
InventoryDialog
ObjectMenu
PauseMenu
PlayerWeaponsDialog

Dialogi

- Czasami nie warto tworzyć nowego activity
- Nie wymagają tworzenia nowej klasy
- Małe, wyskakujące okno (choć nie zawsze!)
- W naszym projekcie używane do interakcji z obiektami i postaciami
- Mogą korzystać z gotowych layouts (pliki .xml)

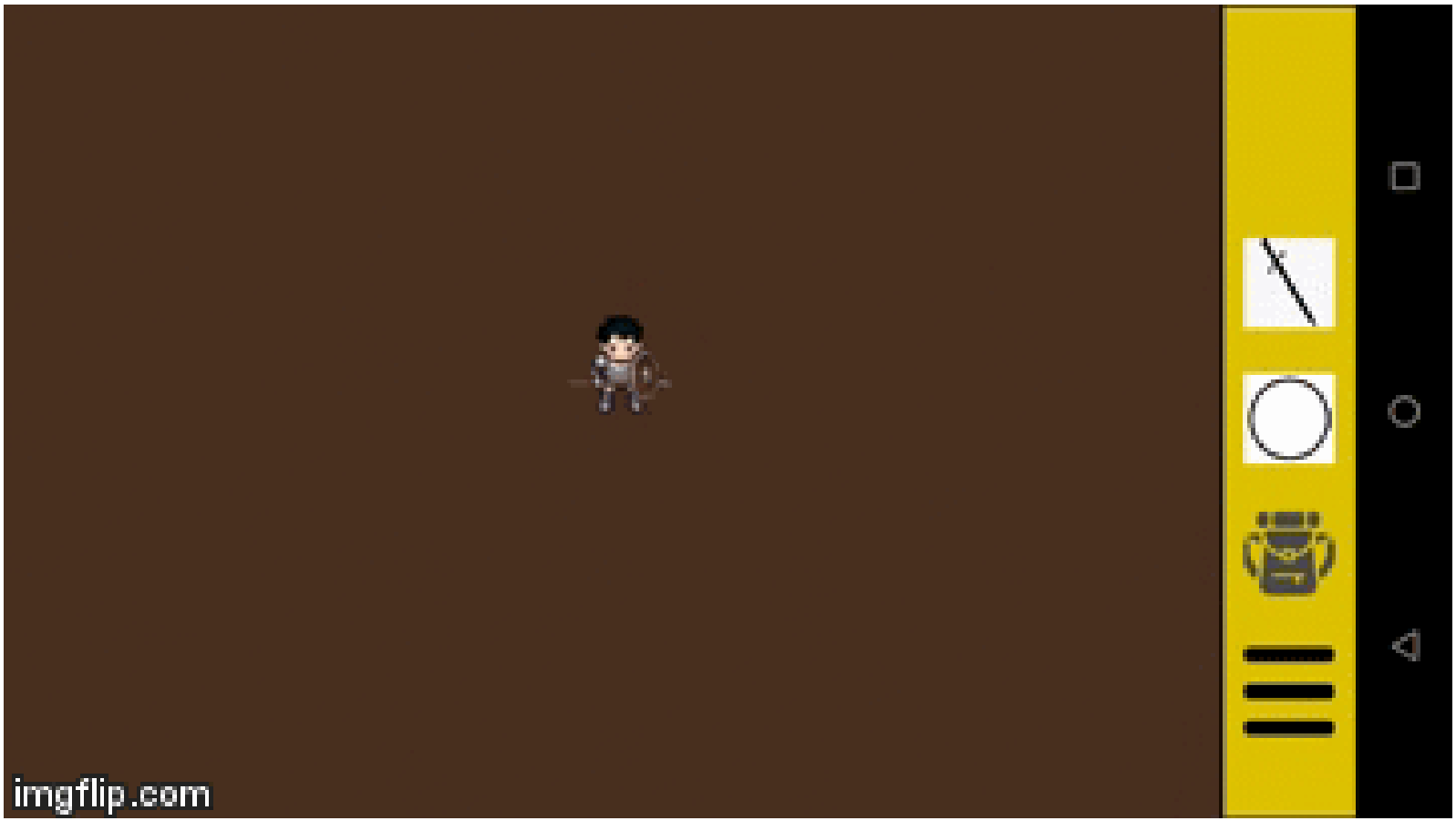
Downolność w tworzeniu dialagów



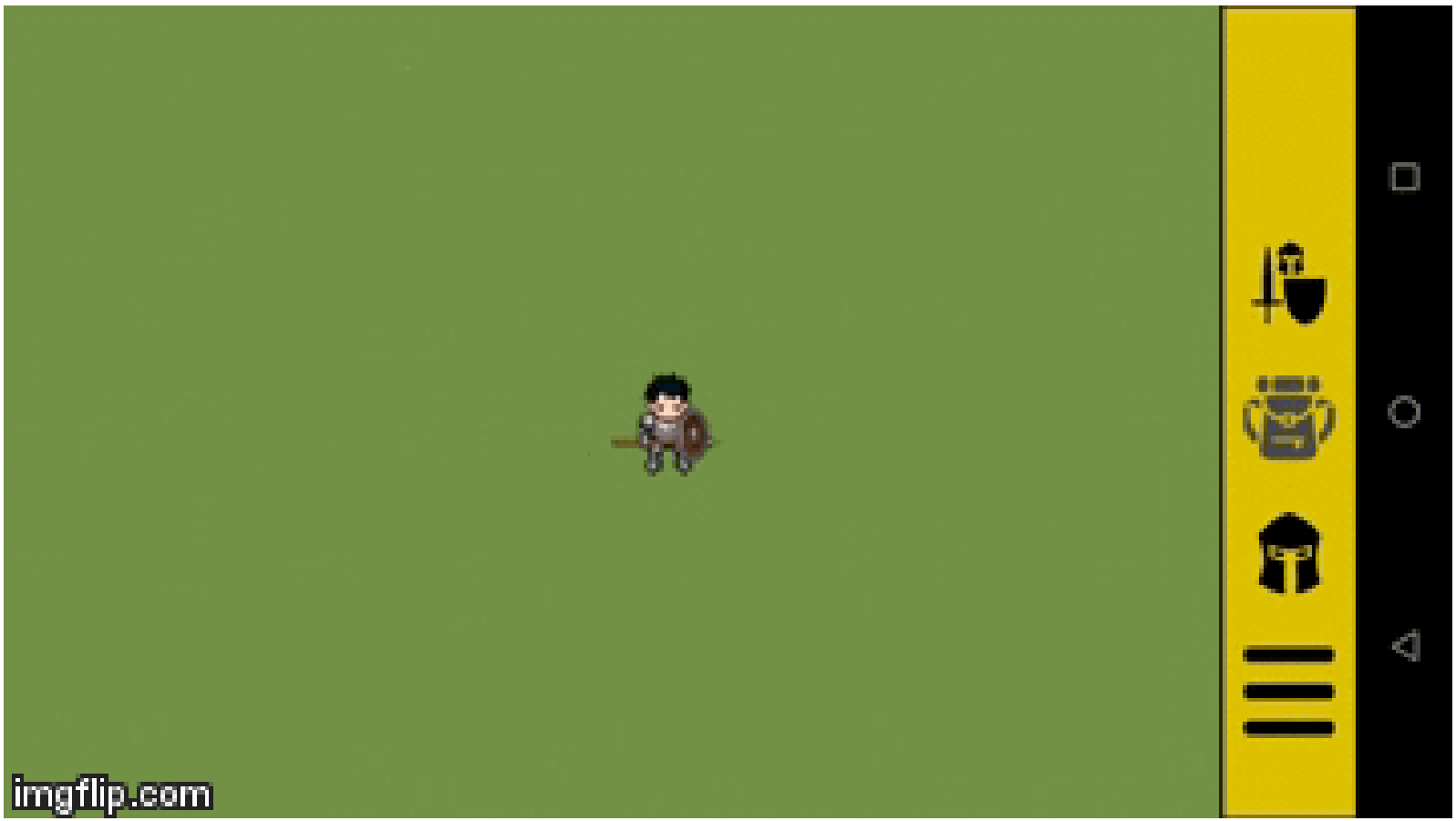
Subklasy dialogów

- AlertDialog
- Date(Time)PickerDialog

Różne sposoby obserwowania ruchu postaci



Różne sposoby obserwowania ruchu postaci



Rysowanie obiektów

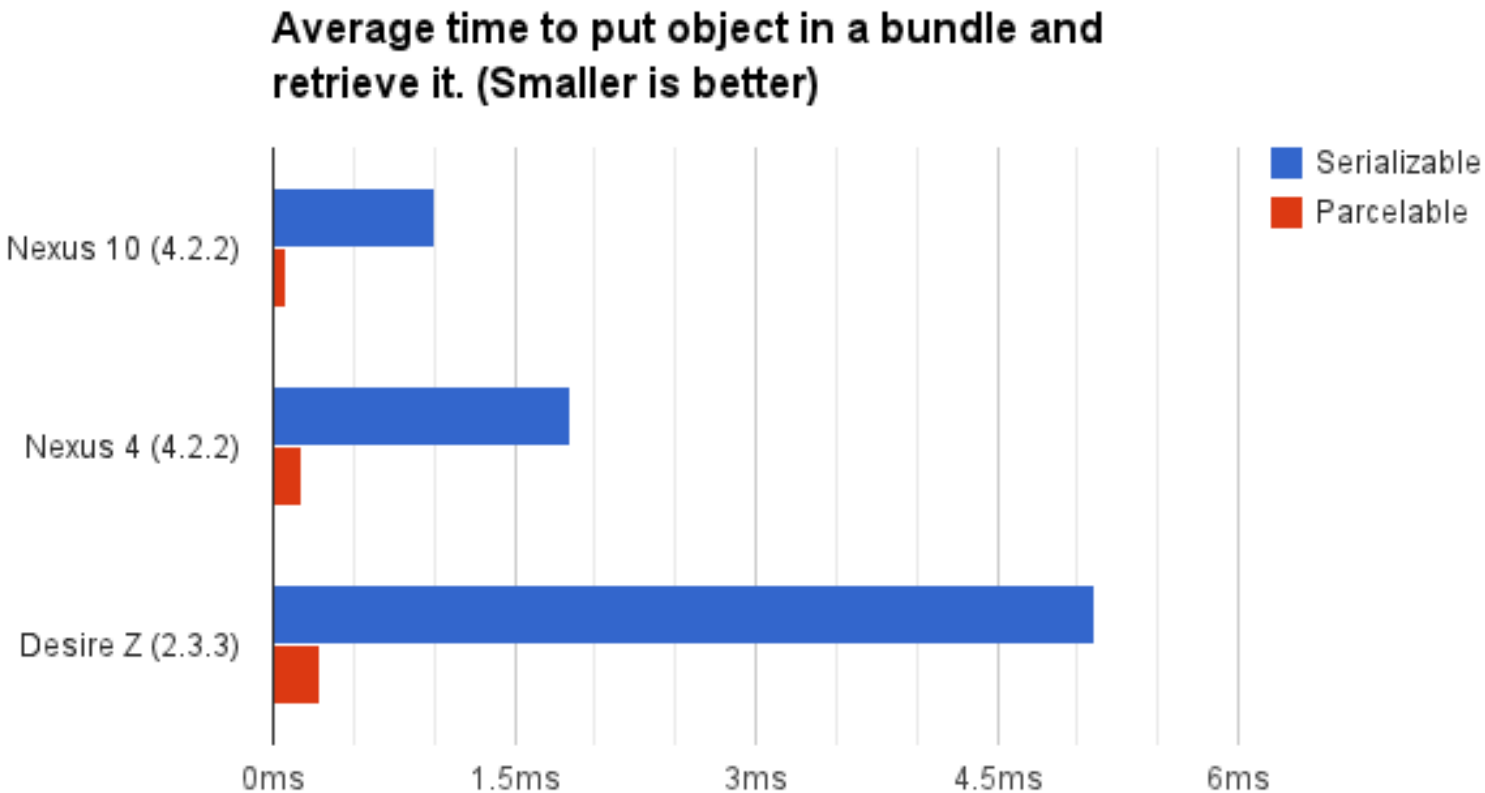
- Klasa Background - "przesuwa" w odpowiedni sposób tło
- Klasa GameObject - przechowuje koordyntaty obietków
- canvas.drawBitMap(obraz.png,x,y) - funkcja do rysowania
- W klasie GameView na podstawie dotknięcia ekranu jest obliczana różnica między statrymi a nowymi koordynatami

Klasa Player - wybrane elementy

- Reprezentuje naszego gracza
- Dziedziczy po GameObject
- Implementuje interfejsy Combat, Parcelable
- Posiada własną animacje
- equip(Item item), unequip(Weapon.WeaponType wt)

Parcelable, czyli lepszy Serializable

- Serializable, czyli piękno tkwi w prostocie
- Parcelable, czyli prostota nie zawsze idzie w parze z wydajnością



Kilka słów o animacji

- Sprite i SpriteSheet
- animationUpdate()
- Przy każdym wywołaniu update(), jeżeli gracz się porusza to przejdź do następnej klatki

Poziomy i statystyki

- 4 główne atrybuty - siła, zręczność, witalność i mądrość
- Na początek 5 punktów do wydania
- 1 lvl = 1 punkt

Przedmioty

- Klasa abstrakcyjna Item
- Każdy przedmiot ma swoje ID, nazwę, oraz bitmape
- Klasa Inventory do zarządzania przedmiotami

Gdzie je możemy znaleźć?

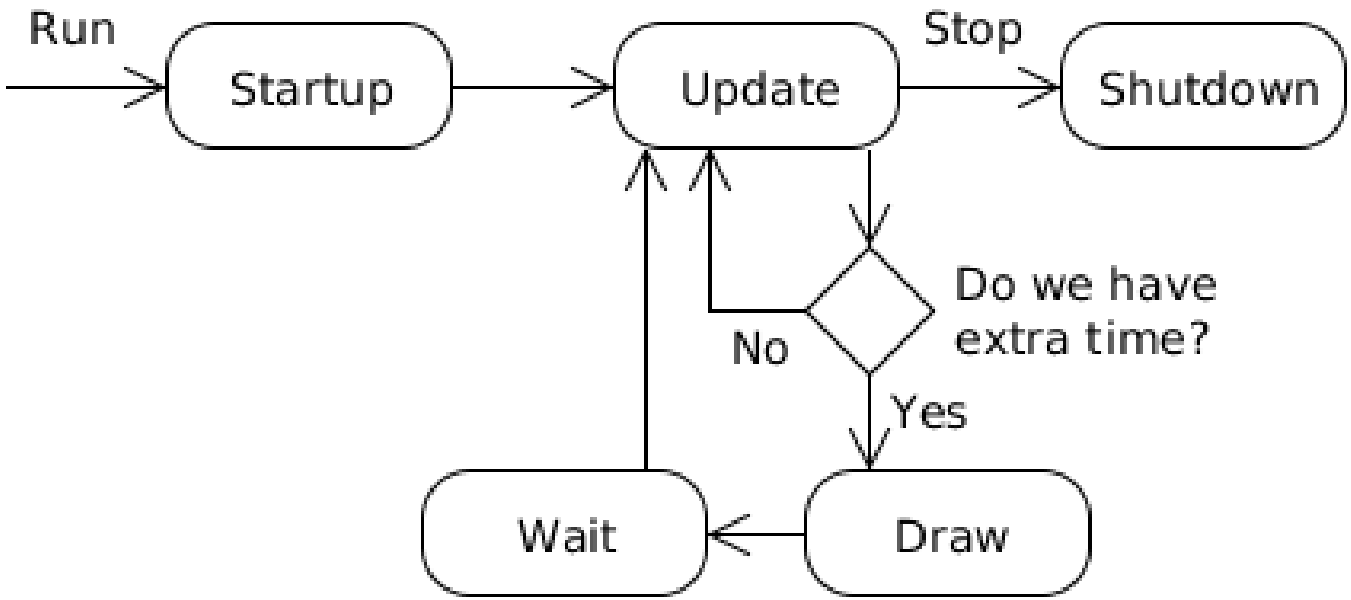
- Niektóre przedmioty dostajemy na start
- Możemy je otrzymać w nagrodę za pokonanie przeciwnika
- Na mapie są skrzynie, w których czekają cenne nagrody

Game Loop, czyli nieskończona pętla...

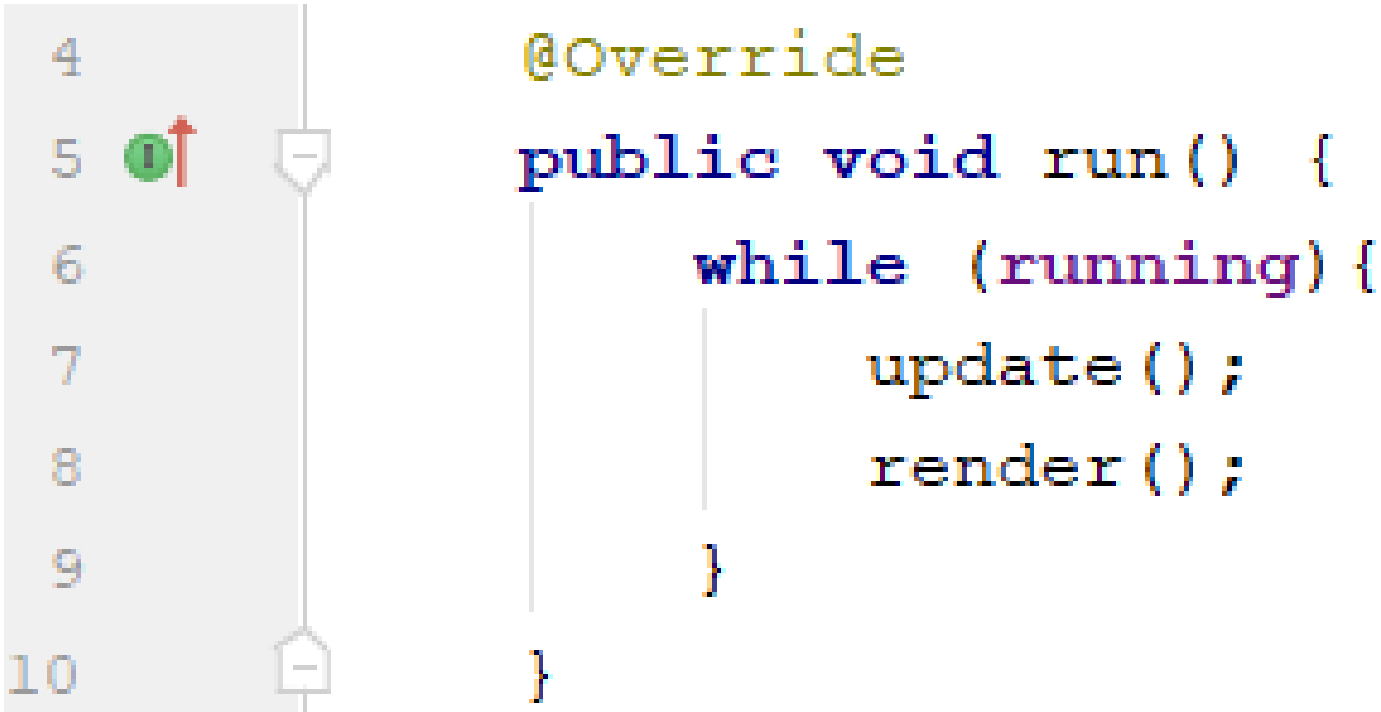
Funkcje:

- Pobranie danych od użytkownika
- Aktualizacja stanu gry
- Renderowanie świata gry

Game Loop, czyli nieskończona pętla...



Game Loop, wersja podstawowa



```
4
5   @Override
6   public void run() {
7       while (running){
8           update();
9           render();
10      }
    }
```

Może i prosta ale bardzo problematyczna

UPS vs FPS

- UPS - Updates per Second, czyli jak szybko nasza gra działa - metoda update()
- FPS - Frames per Second, czyli jak płynnie nasza gra działa - metoda render()

Stała szybkość gry

```
5   public void run() {
6       int DESIRED_FPS = 60;
7       long sleepTime = 1000 / DESIRED_FPS;
8
9       while (running) {
10          update();
11          render();
12          try {
13              Thread.sleep(sleepTime);
14          } catch (InterruptedException e) {
15              e.printStackTrace();
16          }
17      }
18  }
```

Stała szybkość gry - problemy

- Ograniczona ilość FPS-ów
- Thread.sleep() - raczej chcemy unikać
- *sleepTime* - wartość stała, niezależna od update() i render()
- spadek szybkości na wolnej platformie sprzętowej

Zmaksymalizowanie FPS


```
6   public void run(){
7       long lastTime = System.nanoTime();
8       double delta = 0.0;
9       double ns = 1000000000.0 / (float)DESIRED_FPS;
10      while(running) {
11          long now = System.nanoTime();
12          delta += (now - lastTime) / ns;
13          lastTime = now;
14          if (delta >= 1.0) {
15              update();
16              delta--;
17          }
18          render();
19      }
20  }
```

Problem: spadek szybkości na wolnej platformie sprzętowej

Zadania

- Zadania 1 - zadania z GameLoop - w IntelliJ
- Zadania 2 - zadania z ruchem postaci, rysowaniem obiektu oraz edycja pliku .xml - w Android Studio
- https://github.com/miko083/JPWPZadaniaRPG

Do pracy!