

PROJEKT ARDUINO

Technika Mikroprocesorowa

Dominik Łaguń
Maciej Żakiewicz

SPACE RAID

1. na czym polega gra
2. biblioteki
3. hierarchia struktur

1. Na czym polega SPACE RAID

Ogólnie:

Gra typu arcade/przetrwanie z rosnącym poziomem trudności.

Zadaniem gracza jest poruszanie się statkiem kosmicznym, niszczenie lub omijanie losowo generowanych komet.

W określonych odstępach czasowych odbywa się walka z bossem, z każdym poziomem staje się trudniejsza przez rosnącą liczbę bossów i ich wytrzymałość.

W trakcie gry gracz nagradzany jest punktami.

Szczegółowo:

- >Podczas setupu gra inicjalizuje wyświetlacz, przyciski i generator liczb losowych. Wczytuje zapisane wyniki z pamięci EEPROM i resetowane wszystkie obiekty.
- >Gracz steruje statkiem poziomo i może strzelać pociskami w górę ekranu. Komety spadają z losową pozycją, rozmiarem i prędkością, a ich liczba i tempo rosną z czasem.
- >Funkcja rectHit sprawdza kolizje między statkiem, pociskami i obiektami w grze.
- >Kolizja statku z kometą zmniejsza liczbę żyć gracza, zderzenie z bossem kończy grę.
- >Bossowie poruszają się poziomo, odbijając się od krawędzi ekranu i stopniowo zniżają w dół, również używają pocisków. Gra przechodzi między fazą komet a walką z bossami, podnosi poziom trudności.
- >Gra obsługuje pauzę, ekran końca gry oraz zapis i aktualizację rekordu punktowego. Gra przechowuje informacje o punktach również po wyłączeniu arduino.

2. Wykorzystywane biblioteki

2.1

SPI – odpowiada za komunikację SPI między arduino i wyświetlaczem.

2.2

Wire - odpowiada za komunikację I²C między arduino a wyświetlaczem.

2.3

EEPROM – umożliwia zapis i odczyt danych w pamięci trwałej arduino.

W tym projekcie odpowiada za przechowywanie rekordu (highscore).

2.4

Adafruit_GFX – biblioteka graficzna.

Pozwala rysować linie, prostokąty, tekst, punkty itp. na ekranie.

2.5

Adafruit_SH110X – sterownik wyświetlacza SH110X.

Działa na podstawie Adafruit_GFX. Komunikuje się przez SPI/I²C, żeby faktycznie wyświetlać obraz na ekranie.

3 Struktury

Zależność danych elementów kodu zaprezentowano w formie wykresów-drzew.

Ze względu na czytelność, najważniejsze aspekty zostaną rozwinięte w osobnych podpunktach.

Podpunkty zostały oznaczone następująco:

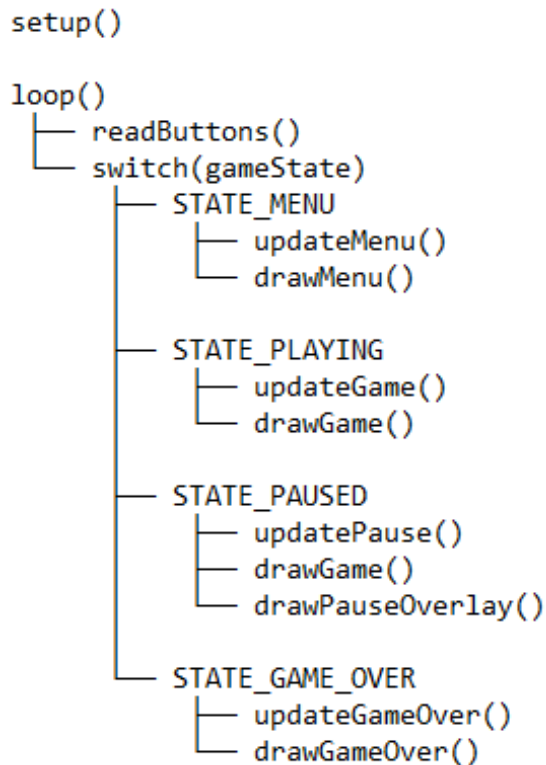
3.1 - Ogólny podział

3.2 - Rozwinięcie: Update Game

3.3 - Całość drzewka

3.1.

Najogólniejszy schemat gry. Składa się z setupu oraz pętli, w której odbywa się gra.

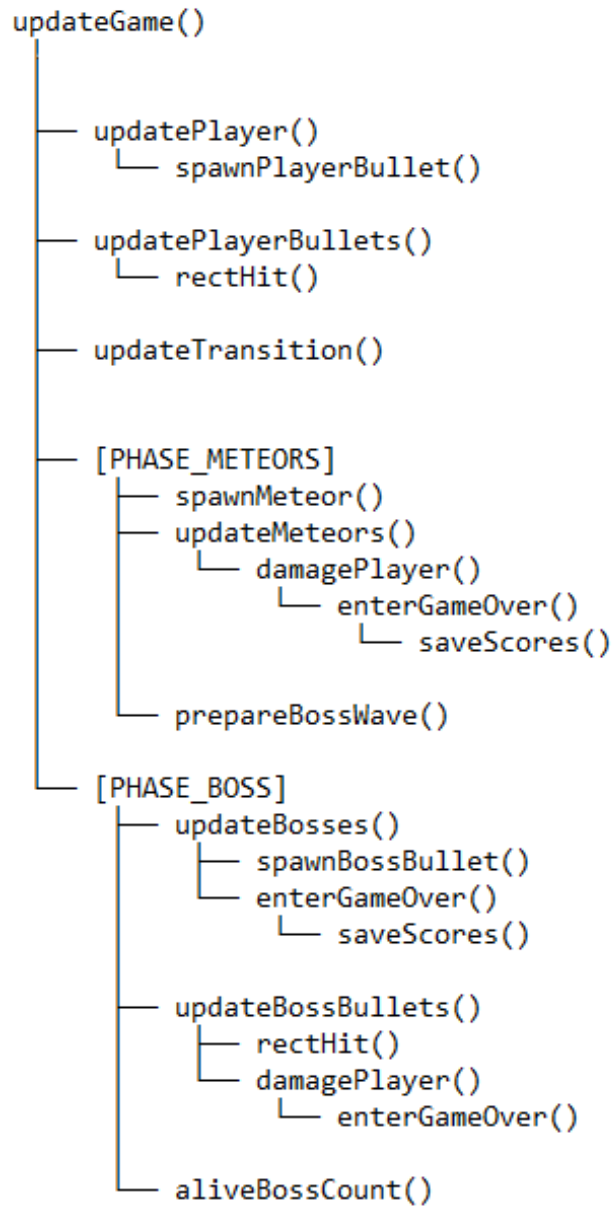


3.2.

Logika gry.

W tym drzewku interesują nas głównie tury, na które dzieli się gra, tj. fazy komet i fazy bossów.

Przedmiotem zainteresowania jest również aktualizacja danych odnośnie gracza, komet, bossów i komunikatów wyświetlających się na ekranie. (np. GAME OVER lub komunikat o wejściu w fazę bossów)



3.3

Poniżej przedstawiono całe drzewko funkcji.

main	
├ init()	inicjalizacja wyświetlacza, przycisków, EEPROM
├ loadHighscore()	wczytanie najlepszego wyniku
├ resetGame()	reset wszystkich obiektów gry
└ loop()	
├ readInputs()	odczyt przycisków sterujących
└ updateGame()	logika gry

└ movePlayer()	ruch gracza
└ moveProjectiles()	ruch pocisków
└ moveComets()	ruch komet
└ spawnComets()	generowanie nowych komet
└ spawnBoss()	generowanie bossa
└ moveBoss()	ruch bossa
└ bossShoot()	strzały bossa
└ rectHit()	sprawdzenie kolizji (statki, pociski, komety, boss)
└ updateScore()	aktualizacja wyniku i życia
└ renderGame()	rysowanie wszystkich elementów na ekranie
└ drawPlayer()	
└ drawComets()	
└ drawProjectiles()	
└ drawBoss()	
└ checkGameOver()	sprawdzenie końca gry
└ saveHighscore()	zapis najlepszego wyniku