Sprawozdanie projektowe z przedmiotu Programowanie Komputerów 4 (C++)

Aleksander Grzybowski, INF sem. 4 gr. 1 sek. PS

1. Temat projektu

SpaceArcade - kosmiczna strzelanka 2D w stylu Space Invaders

1.1 Założenia projektu / spec

- prosta gra zręcznościowa
- tryb graficzny, tekstury, sprite'y
- rosnący z czasem poziom trudności gry
- obsługa za pomocą klawiatury, myszki nie potrzebujemy
- prosty zapis najlepszego wyniku
- czystość designu i poziom obiektowości ważniejsze niż prostota wykonania i łatwość implementacji (projekt zaliczeniowy, nie praktyka)
- przenośność między platformami Win / Lin
- wykorzystanie darmowych bibliotek

2. Analiza

2.1 Algorytmy

W projekcie nie zastosowałem praktycznie żadnych algorytmów, poza prostą detekcją kolizji na podstawie położenia obrysów obiektów. Najwięcej logiki zawarte jest w częściach odpowiedzialnych za 'obliczanie następnej ramki' tzn. wyliczanie pozycji pocisków i przeciwników, detekcja zestrzeleń, sprawdzanie granic dozwolonych położeń obiektów itp.

2.2 Biblioteki

Do realizacji graficznej użyłem SFML2, ponieważ wśród innych rozwiązań jako jedyny zapewniał wsparcie C++ oraz wspaniałą dokumentację w postaci licznych tutoriali na stronie domowej projektu. Ważna jest także wolna licencja biblioteki. Rozważałem użycie Allegro lub SDL. Niestety, Allegro to czyste C i nie można w nim manipulować obiektami reprezentującymi logiczne byty (typu np. 'przesuń coś 10px w lewo'). SDL wydał mi się zbyt skomplikowany i nieczytelny, sam fakt, że nie ma w nim wbudowanej funkcji do rysowania pojedynczych pikseli daje do myślenia.

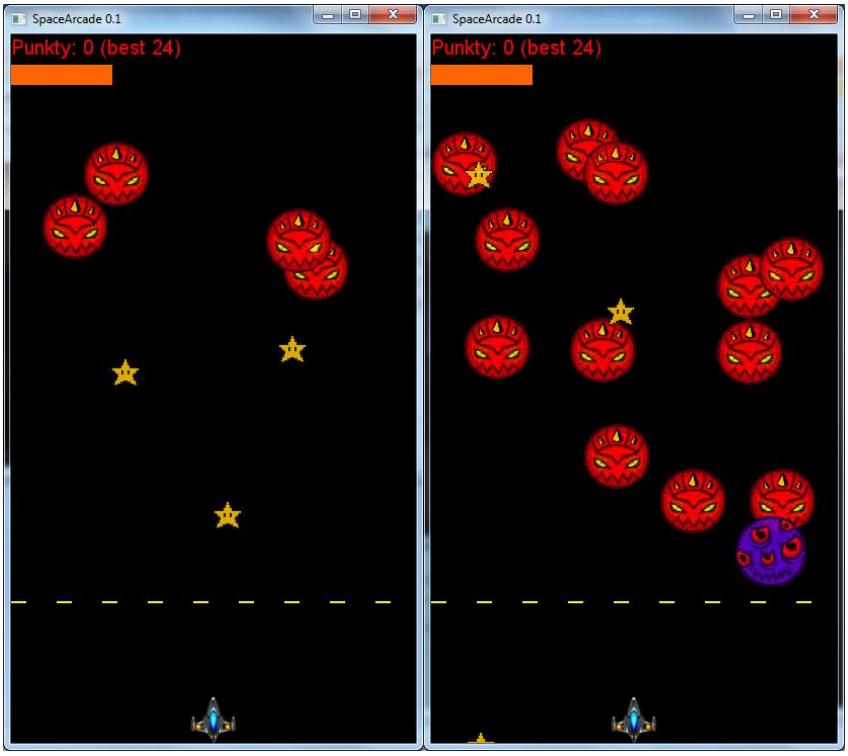
2.3 Decyzje i problemy

Największym problemem okazało się zaplanowanie całej struktury programu, ponieważ nie miałem wcześniej żadnego doświadczenia w tworzeniu gier innych niż kółko i krzyżyk. Zdecydowałem się więc rozpocząć prace od tworzenia byle to działającego prototypu, za każdym razem na boku, w osobnym projekcie testując nowo poznane rozwiązania. Poza tym, postanowiłem wypróbować zasłyszaną gdzieś zasadę, że jeśli kod nie będzie ciągle zmieniany, to w końcu stanie się twardy i nie do ruszenia. Okazało się, że rozwiązanie to na ogół sprawdza się i dzięki temu powstały kod jest łatwo rozszerzalny i czytelny zarazem.

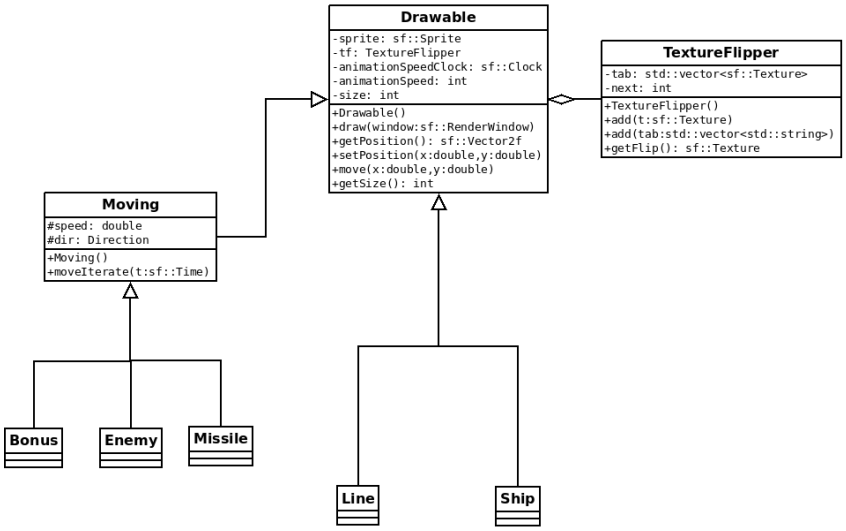
3. Specyfikacja zewnętrzna

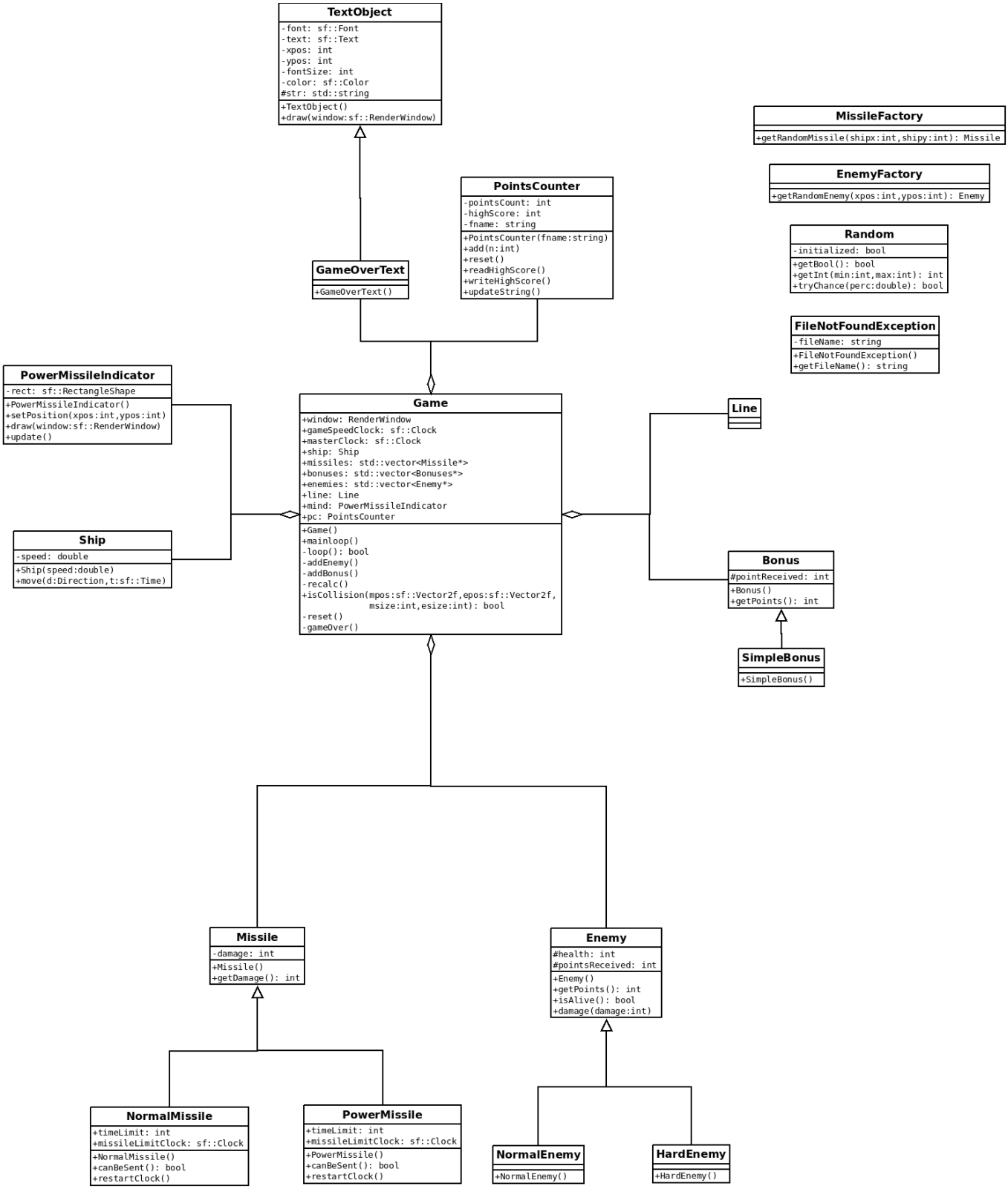
SpaceArcade po uruchomieniu od razu rozpoczyna nową grę. Jeśli wystąpi brak potrzebnych plików (grafik/czcionek), wyrzucony zostanie odpowiedni komunikat na konsolę i program zostaje zakończony. Statkiem steruje się korzystając ze strzałek, a pociski wystrzeliwuje się klawiszami Z oraz X. Klawisz Q zamyka grę. Pozostawiłem deweloperską opcję: klawisz R resetuje grę.

Gra polega na zestrzeliwaniu przeciwników i zbieraniu bonusów. Za powyższe dostaje się punkty. Gra kończy się w momencie, gdy pierwszy przeciwnik spotka się z linią ograniczającą. Do dyspozycji gracza są 2 pociski, zwykły (pod Z) i mocniejszy (pod X). Pociski zwykłe mogą być wystrzeliwane dużo częściej niż mocne. Dla wygody użytkownika, na ekranie obecny jest wskaźnik, demonstrujący upływ czasu do kolejnej możliwości wystrzelenia.



4. Diagram klas




TextObject
-font: sf::Font
-text: sf::Text
-xpos: int
-ypos: int
-fontSize: int
-color: sf::Color
#str: std::string
+TextObject()
+draw(window:sf::RenderWindow)
MissileFactory
+getRandomMissile(shipx:int,shipy:int): Missile
EnemyFactory
+getRandomEnemy(xpos:int,ypos:int): Enemy
PointsCounter
-pointsCount: int
-highScore: int
-fname: string
+PointsCounter(fname:string)
+add(n:int)
+reset()
+readHighScore()
+writeHighScore()
+updateString()
Random
-initialized: bool
+getBool(): bool
+getInt(min:int,max:int): int
+tryChance(perc:double): bool
GameOverText
+GameOverText()
FileNotFoundException
-fileName: string
+FileNotFoundException()
+getFileName(): string
PowerMissileIndicator
-rect: sf::RectangleShape
+PowerMissileIndicator()
+setPosition(xpos:int,ypos:int)
+draw(window:sf::RenderWindow)
+update()
Game
+window: RenderWindow
+gameSpeedClock: sf::Clock
+masterClock: sf::Clock
+ship: Ship
+missiles: std::vector<Missile*>
+bonuses: std::vector<Bonuses*>
+enemies: std::vector<Enemy*>
+line: Line
+mind: PowerMissileIndicator
+pc: PointsCounter
+Game()
+mainloop()
-loop(): bool
-addEnemy()
-addBonus()
-recalc()
+isCollision(mpos:sf::Vector2f,epos:sf::Vector2f,
msize:int,esize:int): bool
-reset()
-gameOver()
Line
Ship
-speed: double
+Ship(speed:double)
+move(d:Direction,t:sf::Time)
Bonus
#pointReceived: int
+Bonus()
+getPoints(): int
SimpleBonus
+SimpleBonus()
Missile
-damage: int
+Missile()
+getDamage(): int
Enemy
#health: int
#pointsReceived: int
+Enemy()
+getPoints(): int
+isAlive(): bool
+damage(damage:int)
NormalMissile
+timeLimit: int
+missileLimitClock: sf::Clock
+NormalMissile()
+canBeSent(): bool
+restartClock()
PowerMissile
+timeLimit: int
+missileLimitClock: sf::Clock
+PowerMissile()
+canBeSent(): bool
+restartClock()
NormalEnemy
+NormalEnemy()
HardEnemy
+HardEnemy()

5. Specyfikacja wewnętrzna

5.1 Elementy widoczne

Game

Główna klasa odpowiadająca za akcje dziejące się na polu gry.

- Game() - konstruktor inicjujący ekran oraz statek
- mainloop() - funkcja odpalająca cyklicznie loop()
- loop() - pojedyncza iteracja gry
- recalc() - najważniejsza metoda, odpowiada za sprawdzanie wszelkich kolizji (statek-linia, przeciwnik-linia, pocisk-krawędź, bonus-krawędź)
- addEnemy() i addBonus() - dodanie odpowiednich obiektów na ekran w odpowiednich miejscach
- isCollision() - pomocnicza funkcja sprawdzająca kolizje między prostokątami
- reset() - przywraca stan początkowy, taki jaki występuje tuż po stworzeniu Game
- gameOver() - sekwencja końca gry, zatrzymanie rozgrywki, wyświetlenie napisu 'Game Over', aktualizacja najlepszego wyniku i zresetowanie gry do stanu początkowego
- @ window - główne okno SFML
- @ gameSpeedClock, masterClock - zegary odpowiadające za szybkość gry oraz jej przebieg
- @ ship - statek
- @ vector<Missile*>, vector<Enemy*>, vector<Bonus*> - kontenery na obiekty wyświetlane i przetwarzane przez grę
- @ line - linia ograniczająca
- @ mind - element pokazujący, czy można wysłać mocny pocisk
- @ pc - licznik punktów, obsługuje też zapis najlepszego wyniku

Ship

Reprezentuje statek poruszany przez gracza. Statek porusza się w granicach określonych w config.h, linia ograniczająca jest ustawiana tak samo. Po kontakcie z bonusem znika on i naliczane są punkty. Kontakt z przeciwnikiem nie ma znaczenia.

- Ship() - konstruktor
- move() - przesuwa statek zgodnie z wciśniętą strzałką
- @ speed - szybkość poruszania się statku

Missile

Reprezentuje pocisk wystrzeliwany przez statek, po kontakcie pocisku z przeciwnikiem pocisk znika a przeciwnik ponosi obrażenia, znikając jeśli stracił wszystkie punkty życia.

- Missile() - konstruktor, przyjmujący na wejście aktualną pozycję statku w celu dokładnego obliczenia pozycji pocisku (środek statku minus połowa szerokości pocisku)
- getDamage() - getter dla damage
- @ damage - ilość zadawanych przez pocisk obrażeń

Enemy

Klasa przeciwnika, porusza się on w dół ekranu ze stałą prędkością, nie ma ograniczeń na nachodzenie przeciwników na siebie.

- Enemy() - konstruktor
- getPoints() - getter dla pointsReceived
- isAlive() - sprawdza czy przeciwnik ma jakieś punkty życia
- damage() - zadaje obrażenia
- @ health - aktualne zdrowie
- @ pointsReceived - ilość punktów zdobywanych za strącenie przeciwnika

Bonus

Klasa bonusu, w aktualnej wersji ma tylko jedną implementację SimpleBonus, po kontakcie ze statkiem bonus znika i naliczane są punkty.

- Bonus() - konstruktor
- getPoints() - getter dla pointsReceived
- @ pointsReceived - liczba punktów zdobywanych za zebranie bonusa

Line

Statyczna grafika reprezentująca linię ograniczającą ruch statku.

- Line() - konstruktor

NormalMissile, PowerMissile

Implementacje Missile, różnią się tylko wartościami w konstruktorach, definiującymi siłę rażenia, sprite, szybkość itd.

- (Normal/Power)Missile() - konstruktor
- canBeSent() - statyczna metoda, sprawdzająca statyczny timer dla każdego pocisku i ustalająca, czy można wysłać następny
- restartClock() - restartuje statyczny zegar ograniczeń
- @ timeLimit - limit czasu pomiędzy wystrzeleniami
- @ missileLimitClock - timer

NormalEnemy, HardEnemy

Implementacje Enemy, różnią się tylko wartościami w konstruktorach, definiującymi początkową ilość punktów życia, prędkość itp.

- (Normal/Hard)Enemy() - konstruktor

SimpleBonus

Jedyna implementacja Bonus w postaci gwiazdki z Mario.

- SimpleBonus() - konstruktor

GameOverText

Służy do wyświetlania napisu oznajmującego koniec gry.

- GameOverText() - konstruktor

PointsCounter

Klasa odpowiedzialna za liczenie punktów, ich ciągłe wyświetlanie, aktualizację oraz ewentualny zapis do pliku.

- PointsCounter() - konstruktor
- add() - dodaje wartość do sumy
- reset() - zerowanie
- updateString() - aktualizacja ciągu znaków wyświetlanego w grze
- writeHighScore() / readHighScore() - operacje na pliku z zapisem
- @ pointsCount - liczba zdobytych aktualnie punktów
- @ highScore - najwyższy jak dotąd wynik
- @ fname - nazwa pliku z zapisanym najlepszym wynikiem

PowerMissileIndicator

Zwykły kolorowy prostokąt, obrazujący stan 'naładowania' mocnego pocisku, jeśli można wystrzelić to jest on pomarańczowy, tuż po wystrzeleniu staje się czarny i z czasem płynnie przechodzi znów w pomarańczowy.

- * PowerMissileIndicator() - konstruktor
- * setPosition() - ustalenie pozycji na ekranie
- * draw() - rysowanie prostokąta
- * update() - aktualizacja koloru
- @ rect - prostokąt z wypełnieniem z SFML

5.2 Elementy niewidoczne

TextureFlipper

Własna klasa do obsługi prostych animacji wieloklatkowych. Zawiera timer, który po przepełnieniu powoduje wzięcie kolejnej klatki i ustawienie jej jako aktualnej.

- TextureFlipper() - konstruktor
- add() - dodawanie tekstur bezpośrednio jako obiekt SFML lub też jako wektor nazw plików
- getFlip() - przejście na następną teksturę i zwrócenie referencji do niej

- @ vector<sf::Texture> - tekstury
- @ next - numer następnej tekstury do wyświetlenia, po ostatniej nadchodzi pierwsza

Drawable

Własna klasa obsługująca obiekty wyświetlane na ekranie. Enkapsuluje większość bezpośrednich wywołań rysowania czy też zmiany pozycji, i przekazuje je bezpośrednio do obiektu sprite w niej zawartego. Niestety, w celu uproszczenia postanowiłem, że ta klasa posiada już sprite, także elementy nie zawierające ich (np. PowerMissileIndicator) muszą same radzić sobie z wyświetlaniem.

- Drawable() - konstruktor
- draw() - rysowanie obiektu na aktualnej pozycji
- getPosition() / setPosition() / move() - operacje na pozycji
- getSize() - getter dla size
- @ sprite - aktualny sprite
- @ tf - obiekt TextureFlipper, obsługujący animacje
- @ animationSpeedClock - dla każdego obiektu osobny timer animacji
- @ size - rozmiar obiektu

Moving

Częściowa implementacja Drawable, określa obiekty poruszane przez komputer na ekranie.

- Moving() - konstruktor
- moveIterate() - serce mechanizmu poruszania się wszystkich obiektów samoporuszających się, zmienia ich położenie na podstawie czasu który upłynął od ostatniej klatki
- @ speed - szybkość poruszania się
- @ dir - kierunek poruszania się

TextObject

Odpowiada za wyświetlanie tekstu własną czcionką.

- TextObject() - konstruktor
- draw() - rysowanie na aktualnej pozycji
- @ str - ciąg znaków wyświetlany
- @ font - czcionka
- @ text - obiekt SFML umożliwiający tworzenie graficznego tekstu
- @ xpos, ypos - pozycja tekstu
- @ fontSize - wielkość czcionki
- @ color - kolor napisu

MissileFactory, EnemyFactory

Klasy niby-fabryki generujące losowe pociski. Pierwsza, generująca pociski, nie jest używana w wynikowym programie (pozostałość po testach, można by wykorzystać ją do np. losowego wystrzeliwania pocisku za mniejszą

cenę), natomiast druga używana jest do losowego tworzenia przeciwników, przy czym częściej tych słabszych niż silniejszych.

- getRandom(Missile/Enemy) - zwraca nowo utworzony obiekt

Random

Służy do generowania danych losowych, statyczna całkowicie.

- getBool() - zwraca true/false z równym prawdopodobieństwem
- getInt() - zwraca liczbę z podanego jako parametr zakresu
- tryChance() - zwraca true/false w zależności od podanego zakresu procentowego

FileNotFoundException

Klasa-wyjątek informująca o braku potrzebnej grafiki/czcionki itp.

- FileNotFoundException() - konstruktor
- getFileName() - getter dla fileName
- @ fileName - nazwa pliku

6. Przebieg prac i testowanie

Korzystając z okazji, postanowiłem nauczyć się gita. Bardzo spodobało mi się to rozwiązanie, chociaż wykorzystywałem może parę % jego możliwości. Zamieszczam zrzut loga z przebiegu prac.

```
5e19ecb Najlepszy wynik
ea5d3ab Komenty
c1e03da NormalMissile wystarcza 1 strzał żeby pokonać NormalEnemy
d5bf85d Normalne bonusy
5c642e1 Ugrywalnienie, wywalenie nieużywanej addMissile (ślepy jestem)
212a98a Ref, komentarze, consty i &-y
ce07912 Ref
85a4b25 Lepsze nazwy plików sprite
15de1a1 Ref
4d31dd1 Komenty
66e9401 Ref, resetowanie zegara po przeganej
f7bc902 Trudność narasta z czasem (prymitywne)
057bc9c Ref
0f72e4b Ref, prostszy sposób z kolorami
0fbd8c3 ładniejszy kolor wskaźnika powermissile
597cea2 Ref, wywalenie paru stałych
7d4ac21 Ref, zmiana kolejności wyświetlania
a7e2c03 Ref
f87df1d Ref
47d877a Ref
d42804c TextObject i pochodne też przepisane na ctory
90bd2fd Ref
413d6c9 Ref
deee8d3 GIGANTYCZNY refact, zmiana virtuali na pola i ctory
a0eb763 Ref
7ed6b79 Drawable setTexture niepotrzebne wywalone
26646f5 Moving deleguje sprity do Drawable
b3cfb9f Duża zmiana z initialize() na c-tory
d36f8f9 Komentarze / ref
b642e67 Ref
22125a5 Wskaźnik czy można wystrzelić PowerMissile
```

```
6373a57 Statek lepszy sprite!
ba362ac Sprity!
24a47d5 Metoda restartClock zamiast grzebania w Game
73389b8 Parę constów
00da1d5 getSprites zwraca const wektor
049896d Wywalone niepotrzebne getSpritesString
dd13caf sleep -> sf::Time (przenośne)
37a334f Ref
4beee91 Ref
34d9756 Ref
04a8a9a getSpritesString do Drawable
a8efc1e Ref
541ffde Wielkość Enemy stała - brana z config-u
2809907 Pare sprite - fireball-like
5940f12 Sprawdzanie czy pliki istnieją
85e9013 Komentarze, getColor dla tekstów
4b1f0e3 getAnimationSpeed zamiast stałej dla każdego
3225272 Hierarchia TextObject i uproszczenie
833b380 Prosty Makefile + reguły do gitignore
6e46dc9 Refact, random chance jako double
f299359 draw z animacjami do Drawable, naprawiony błąd SIGFPE
760735d animationSpeedClock do Drawable
3130181 TextureFlipper do Drawable - prościej
605725c Refact
80d0467 Poprawki w Drawable
58fc0a5 Refact
4606f7d Poprawiona szybkość wszystkiego
163abe8 Pododawane const-y
24f4c2e Naprawiony bug z wolnymi pociskami
5c52548 Interfejs Moving dziedziczy z Drawable, refact
17f02c7 Interfejs Moving i proste Bonus'y
6ee9d90 Klasa Random upraszczająca losowanie
338f416 Nie ma zmitaka
11b6430 Coś tam na labkach
feebbd0 Statki nie powstają poza ekranem
7c617ec Dodanie priv/prot/pub
92aaec2 Sprite dla linii
80b19b6 Refact
d9704cd Refact
2af40ea Duuużo refact
4885f17 Interfejs Drawable żeby ładnie wyglądało
fe0bd0b Napis Game Over
acaa6cc Poprawione jeszcze raz kolizje z linią
31fadc9 Poprawiona szybkość Enemy i kolizje z linią
b4d4111 Dodano 'wirtualny konstruktor' Missile
f09d544 Refact
282b93f Dodano prosty reset gry
0c199a1 Wzajemne położenie statek-linia poprawione
b08db10 Enemy dają różne punkty
de8ebb2 Przeciwnicy nie są tworzeni blisko prawej krawędzi
de7c918 Refact, zmiana wektorów na pary intów
a8d155f Licznik punktów jako osobna klasa
e00f01a Uproszczone dodawanie do TextureFlipper
90ba8a2 Lepsze do debugu sprity
8ea20a9 Dodano fabrykę pocisków, po co - nie wiem :(
6a4ea0f Dodano fabrykę przeciwników
47f2cf7 Refact
4877767 Refact
a1ec3de Naprawiony bug z Enemy poza ekranem
3193a77 Zmiana! Statki spadają tylko w dół!
77570f6 Dodana linia graniczna
ef6d2d1 Naprawiony gigatntyczny wyciek pamięci
eaaad9f Refact
33abb55 Dodane zabezpieczenie przed realokacją TextureFlipper
edacf03 Lepsze sprite
8063065 Klasa TextureFlipper i proste animacje wieloklatkowe
eefd64e Testowa prosta animacja missile
f456425 Obiekt rysuje się teraz sam
57251be Refact
f369778 Wywalenie niepotrzebnej zależności headerowej
93000d2 Takie tam małe
```

```
92908a0 Zmiana w spricie bez efektu
bac2de6 Zmiana spritów na poprawne politechnicznie
697c697 Więcej komentarzy wywalonych
e483a58 Wywalone komentarze do debugu poprzedniego cmita
e1f291f Naprawiona detekcja krawędzi
3ed11a9 Poprawiona kontrola wychodzenia za ekran (nie działa)
ea1c564 Sprity oznaczone kolorami
92eeadb Refact
f1381bc Folder na czcionki
304f052 Prosty sprite dla enemy
8dda2f3 Liniowe poruszanie się statków bez sprawdzania granic
85ed864 Missile na sprity
84ebcd6 Sprite dla statku dodany, pamiętać o wielkościach
e034360 Wyświetlanie prostych punktów
aab0628 Sprawdzanie kolizji przez sf::Rect#intersects
5378771 Przerobienie wszystkiego na RectangleShape
ba3537e Refactor
2ee6105 Limity czasowe na pociski przeniesione do ich klas
79c4714 Pozycja pocisku obliczana wewnątrz Missile a nie obok
73d6c96 Missile uniezależnienie od config-a
e294865 Przeniesienie char. enemy do ich klas z configa
0d8b09c Statek może się poruszać do góry
a31c119 Działające liczenie damage i usuwanie jeśli nie żyje
6ce4025 missileLimitClock statyczny osobny dla każdej
8389b90 Poprawa kodu, trochę refact
28b56be Podział Enemy, na razie bez specyf. zachowania
212ea3a Przykładowy podział na NormalMissile i PowerMissile (szybsza)
6d97cf8 Podział pocisków: główna klasa Missile i pochodne konkretne
dc2555a Poprawiam stałe przedrostkiem CONF_, statek i pociski definiowane
10f628a Direction, wywalone std, statek ma limity ruchu, pociski poza ekr
e4e964a Testowe zestrzeliwanie przeciwników
5b74b57 Pierwsze koty za płoty, okno ze statkiem, plik config.h z const i
fe14aa3 Puste repo
50a3102 Initial commit
```

Ciężko stwierdzić w sensie inżynierii oprogramowania, czy gra była testowana. Podczas jej tworzenia, a zwłaszcza w końcowych etapach uruchomiłem ją przynajmniej parę tysięcy razy i nie zauważyłem żadnych błędów, a znalezione były zawsze na bieżąco naprawiane. Jestem zwolennikiem TDD, ale pisanie testów jednostkowych do gry zręcznościowej wydawało mi się zbyt niepraktyczne. Żałuję, że nie wprowadziłem chociaż częściowych asercji, które pomogłyby mi wykrywać stany zabronione szybciej, niż wykrywane były one poprzez dziwne segfaulty. Nie pomaga tutaj SFML, dla którego umieszczenie obiektu w ujemnych współrzędnych rzędu -1000 całkowicie poza ekranem nie jest problemem (it's not a bug, it's a feature, ale mogłoby chociaż generować ostrzeżenie w czasie wykonania).

7. Wnioski

Przy tworzeniu projektu nauczyłem się praktycznego użycia biblioteki graficznej SFML i poznałem dobre praktyki wytwarzania oprogramowania. Większość studentów pisze w tym miejscu, że będzie rozwijać projekt dalej i tym podobne. Pomimo tego, że kod przeze mnie napisany jest dobrze przystosowany do modyfikacji i rozszerzania go, spędziłem nad nim tyle czasu, że nie mam już ochoty go widzieć na oczy. Myślę nad jakąś kolejną grą, tym razem dużo większą i ciekawszą.