Dokumentacja projektu z zajęć programowania obiektowego

Gra „AngryJets”

Michał Zimniak

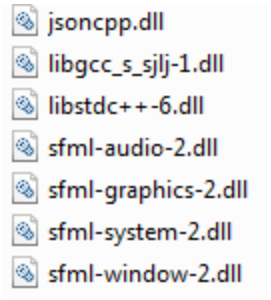
Założenia - Stworzenie gry podobnej do tytułu „Jetmen Revival” ale nieco uproszczonej.

Gra jest prowadzona pomiędzy dwoma graczami na jednym komputerze. Każdy gracz wciela się w rolę pilota uzbrojonego statku kosmicznego i za zadanie ma zniszczenie statku przeciwnika. Gracz musi uważać, żeby się nie zderzać z otoczeniem co skutkuje rozbiciem statku. Gra kończy się gdy jeden z graczy wyczerpie swój limit żyć. O zwycięstwie decyduje zebrana liczba punktów. Za zniszczenie statku przeciwnika dostaje się 1pkt, za zdobycie flagi 5pkt.

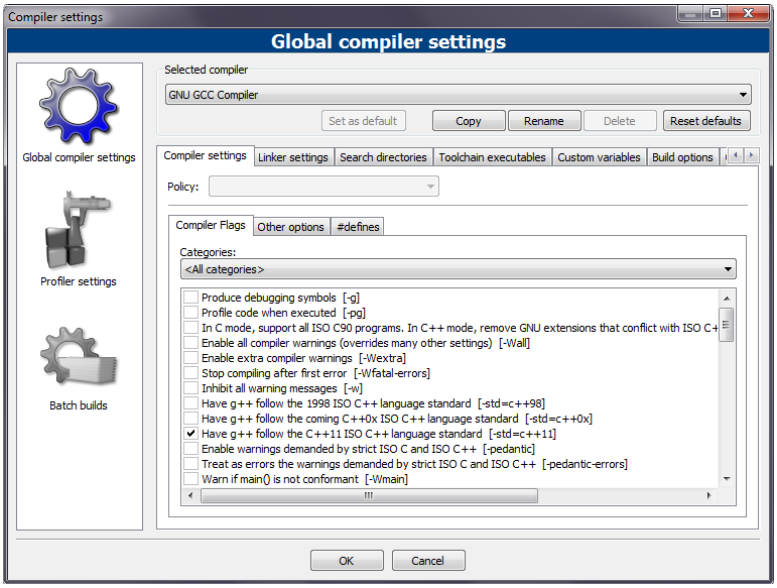
Kompilacja

Gra jest napisana w języku C++ w standardzie C++11. Wymagane jest używanie kompilatora mingw 4.8.1sjlj. W przeciwnym razie nie zadziałają dołączone bliblioteki .dll. Gra korzysta z następujących bibliotek:

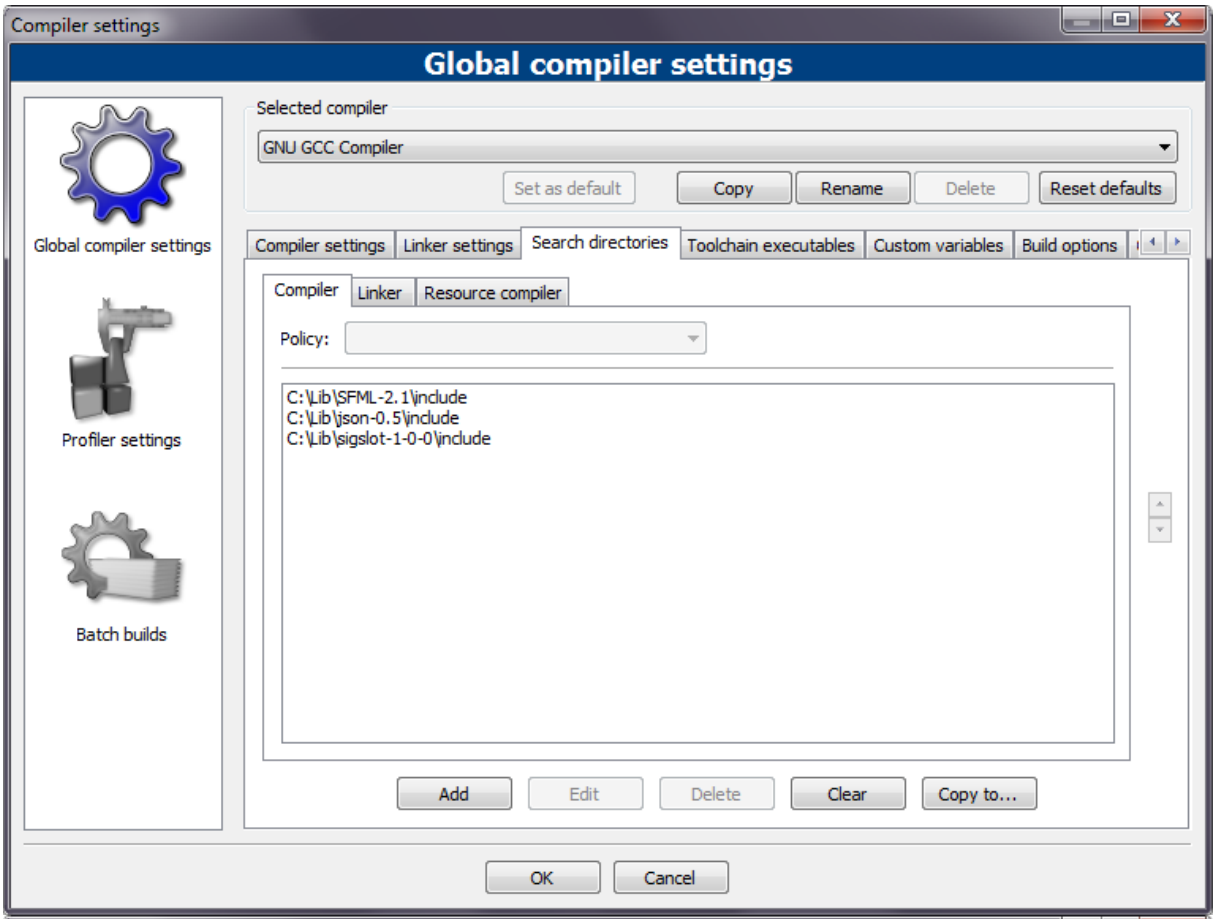
- jsoncpp 0.6.0-rc2
- SFML-2.1
- sigslot-1-0-0

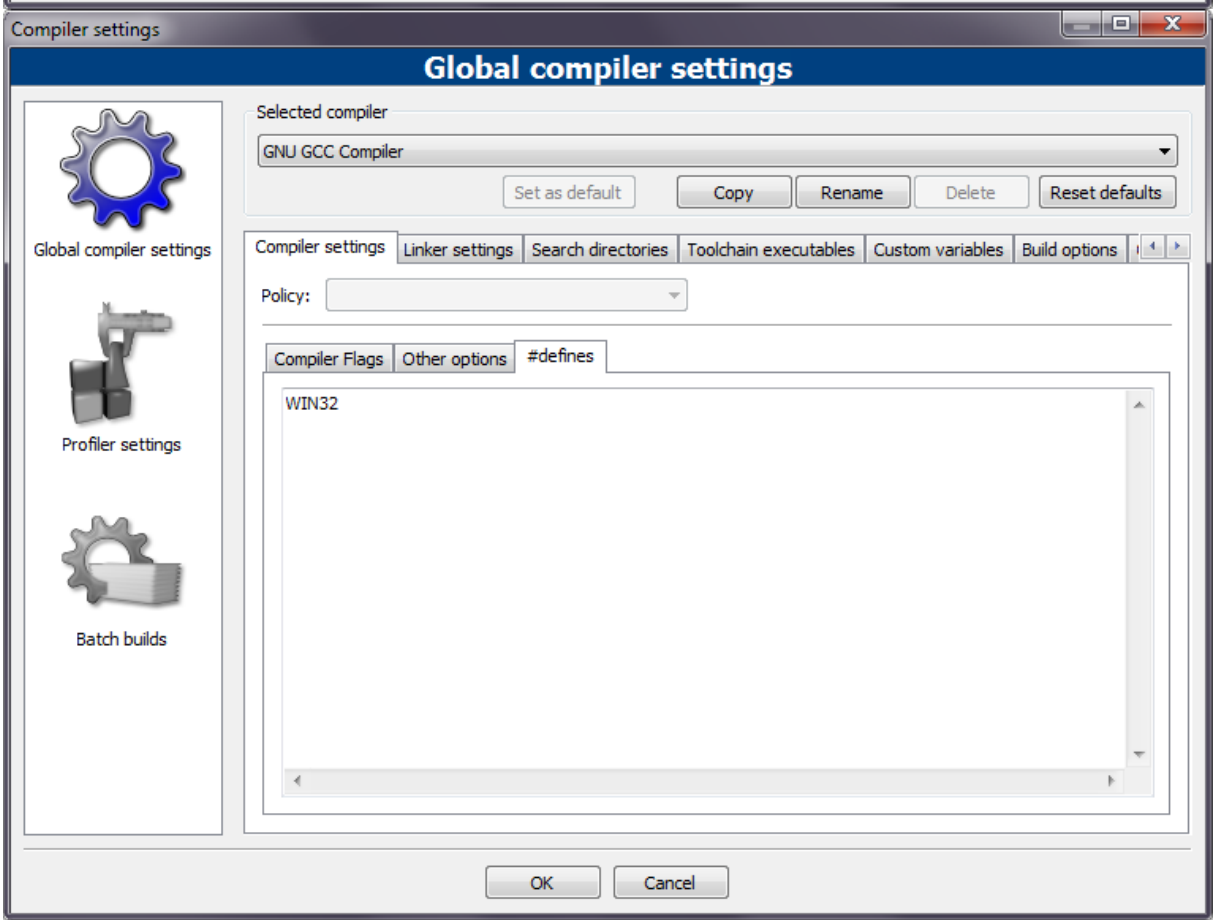


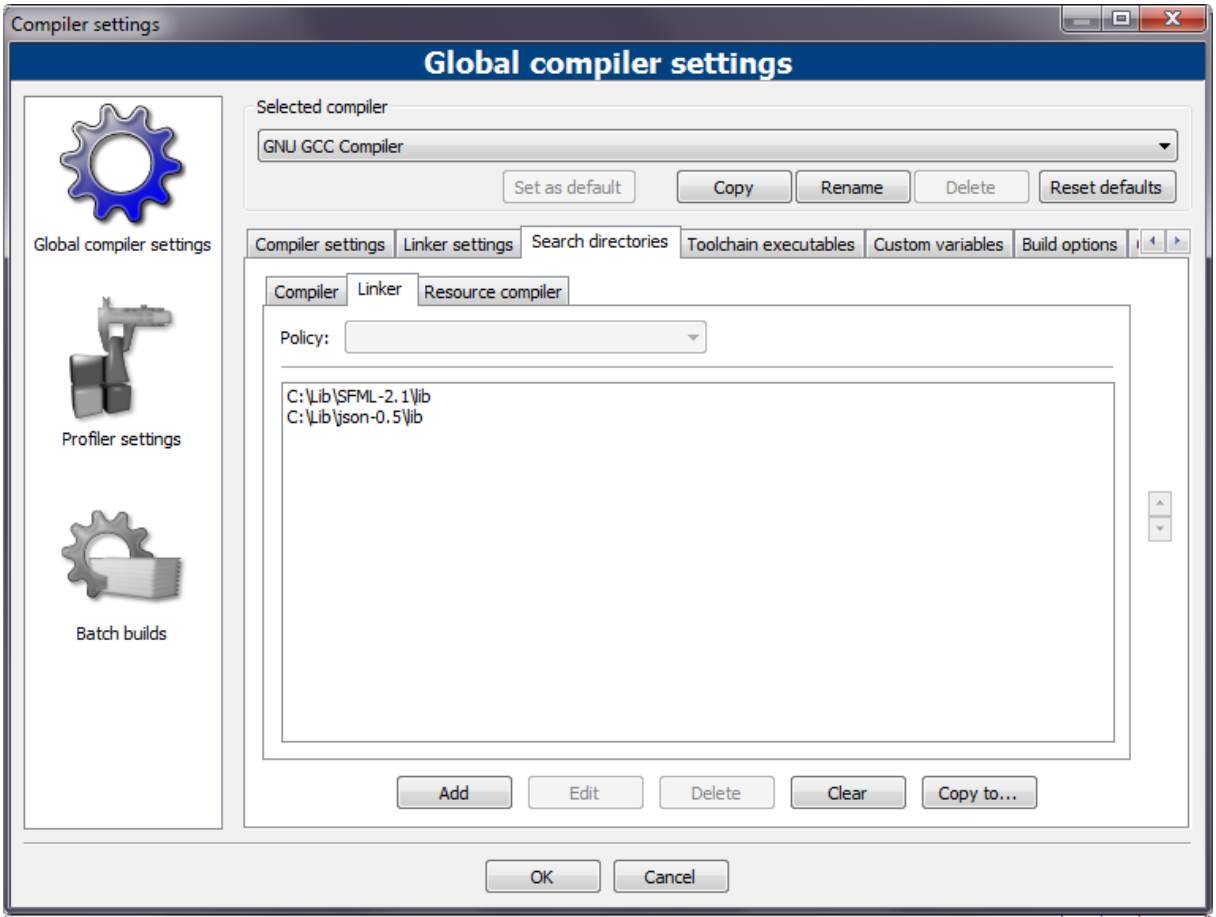
Projekt gry został stworzony za pomocą środowiska Code::Blocks 12.11.
Środowisko zostało skonfigurowane w następujący sposób:

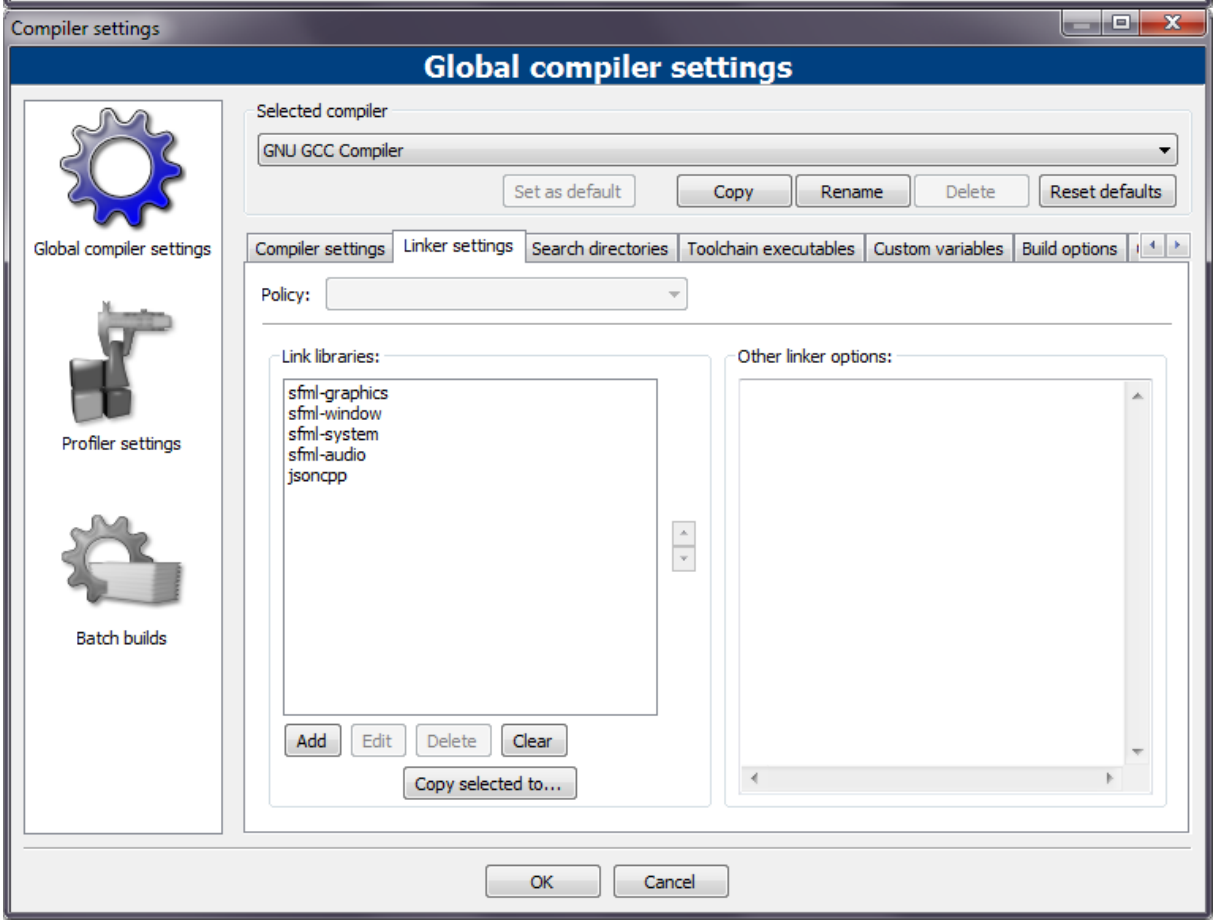


(wybrany standard C++11)


Compiler settings
Global compiler settings
Global compiler settings
Profiler settings
Batch builds
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
Compiler
Linker
Resource compiler
Policy:
C:\Lib\SFML-2.1\include
C:\Lib\json-0.5\include
C:\Lib\sigslot-1-0-0\include
Add
Edit
Delete
Clear
Copy to...
OK
Cancel


Compiler settings
Global compiler settings
Global compiler settings
Profiler settings
Batch builds
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
Policy:
Compiler Flags
Other options
#defines
WIN32
OK
Cancel


Compiler settings
Global compiler settings
Global compiler settings
Profiler settings
Batch builds
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
Compiler
Linker
Resource compiler
Policy:
C:\Lib\SFML-2.1\lib
C:\Lib\json-0.5\lib
Add
Edit
Delete
Clear
Copy to...
OK
Cancel


Compiler settings
Global compiler settings
Global compiler settings
Profiler settings
Batch builds
Selected compiler
GNU GCC Compiler
Set as default
Copy
Rename
Delete
Reset defaults
Compiler settings
Linker settings
Search directories
Toolchain executables
Custom variables
Build options
Policy:
Link libraries:
sfml-graphics
sfml-window
sfml-system
sfml-audio
jsoncpp
Other linker options:
Add
Edit
Delete
Clear
Copy selected to...
OK
Cancel

Analiza obiektowa

Spis klas:

B

BitmaskManager (engine)
Bullet (game)
BulletMinigun (game)

C

Collidable (game)

F

Flag (game)

G

GameLoop (game)
Ground (game)
GuiHighlightedLabel (engine)
GuiHorizontalLayout (engine)
GuiLabel (engine)
GuiLayout (engine)
GuiVerticalLayout (engine)
GuiWidget (engine)
GuiWindow (engine)
Gun (game)
GunMinigun (game)

H

Hud (game)

J

Jetman (game)

N

NonCopyable (engine)

P

Player (game)

R

ResourceManager (engine)

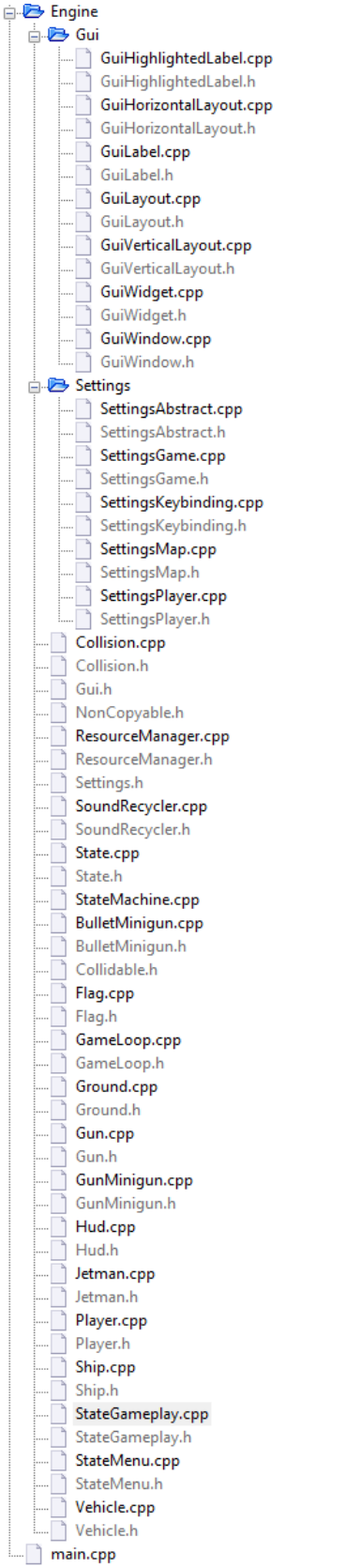
S

SettingsAbstract (engine)
SettingsGame (engine)
SettingsKeybinding (engine)
SettingsMap (engine)
SettingsPlayer (engine)
SettingsPlayerSpawn (engine)
Ship (game)
SoundRecycler (engine)
Speed (engine)
State (engine)
StateGameplay (game)
StateMachine (engine)
StateMenu (game)

V

Vehicle (game)

implementacja architektury



Architektura:

Gra to nic innego jak wielka baza danych przechowująca różne obiekty i wyświetlająca ich stan na ekranie. Klasa GameLoop reprezentuje mechanizm, który sprawia, że wszystko dzieje się w czasie rzeczywistym. GameLoop decyduje jaki stan gry będzie w danym momencie uaktywniony i periodycznie wywołuje jego metodę update(). Stan gry (State) można traktować jako swego rodzaju „minigrę”, która działa na własnych zasadach o ile implementuje interfejs stanu. W „AngryJets” są dwa stany: StateMenu i StateGameplay. Pierwszy wyświetla menu gry, które modyfikuje SettingsGame. StateGameplay przechowuje i wyświetla planszę gry, samoloty, pociski itp. i zarządza całą logiką rozgrywki.

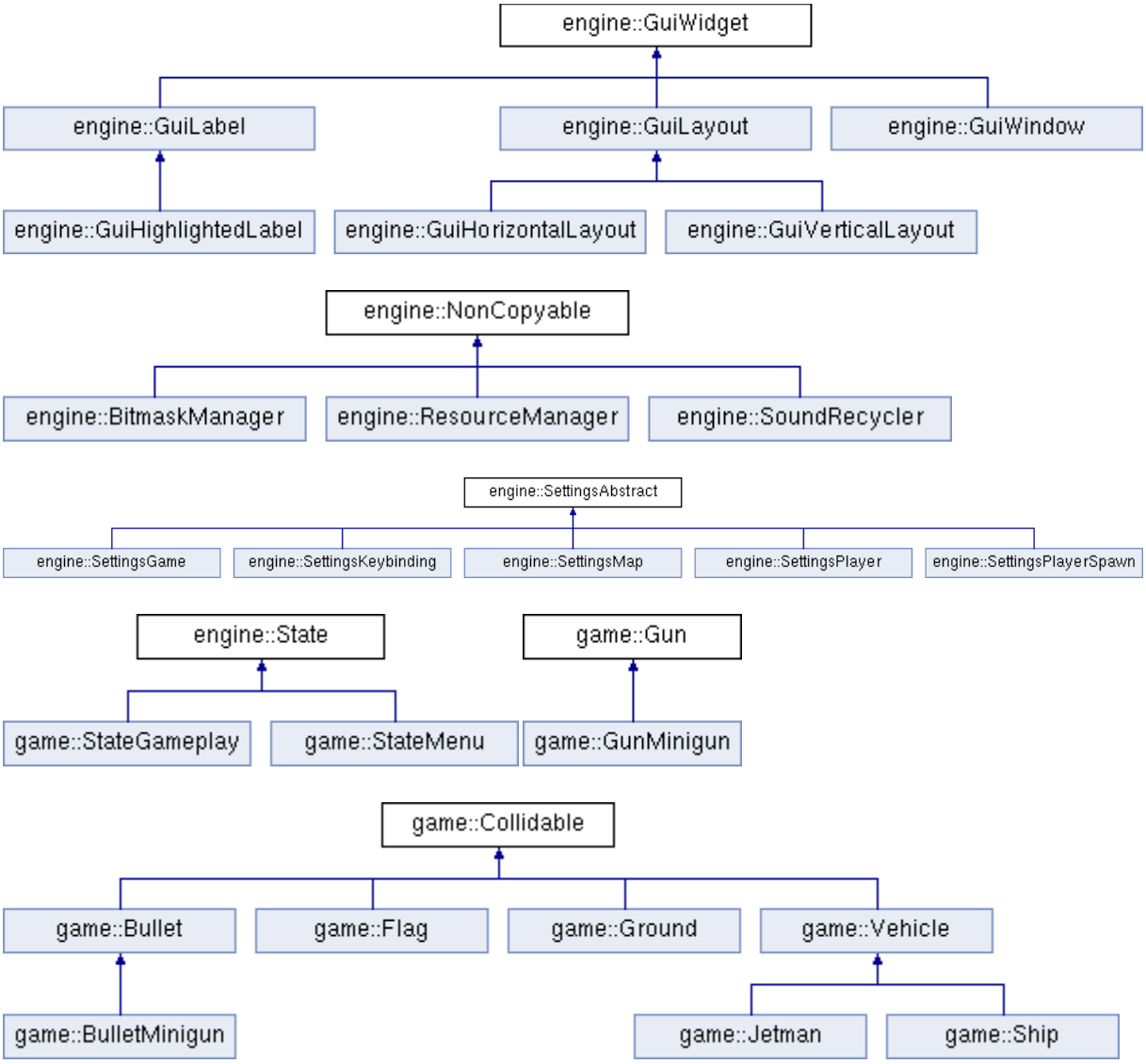
Reszta klas to klasy pomocnicze, które albo reprezentują jakiś obiekt ze świata gry np. Flag, Vehicle, Jetman, Ship,Bullet, Ground, albo jakiś mechanizm wykonujący daną czynność np. BitmaskManager, SoundRecycler...

W projekcie jest też wiele plików zawierających różne pomocnicze funkcje np. do wykrywania kolizji (w końcu C++ jest wieloparadygmatowy).

Wzorce projektowe:

1. **Singleton** (SoundRecycler,BitmaskManager,ResourceManager)
   Np. SoundRecycler jest singletonem bo instnieje fizyczny limit strumieni dźwiękowych. Dzięki wzorcowi singleton mamy zagwarantowane, że jakiś inny SoundRecycler nie korzysta z dodatkowych strumieni.
2. **Mixin** (Gun)
   Klasa dziedzicząca implementuje metodę shootStrategy() i mechanizmy w klasie Gun odpowiednio ją wykorzystują.
3. **Registry** (Settings)
   Zamiast globalnie tworzyć zmienne możemy stworzyć obiekt z publicznymi polami, który udostępnimy tylko w wybranych miejscach.
4. **Flyweight** (ResourceManager w połączeniu z obiektami gry)
   Obiekty gry nie przechowują tekstur, dźwięków czy czcionek. Mają tylko wskaźniki do tych obiektów. Pozwala to zaoszczędzić pamięć.
5. **Signal-Slot**
   Stany gry mogą komunikować się np. z GameLoop bez dokładania zależności pomiędzy klasami

Związki między klasami (wymienione tylko klasy, które coś dziedziczą):



Klasy:

Nazwa	Opis
GuiHighlightedLabel	Etykieta, która po najechaniu myszką się podświetla.
GuiHorizontalLayout	Wyświetla widgety jeden obok drugiego w równych odstępach.
GuiLabel	Zwyczajna etykieta, która wyświetla tekst.
GuiLayout	Abstrakcyjna klasa do przechowywania i wyświetlania widgetów w ustalonym położeniu.
GuiVerticalLayout	Wyświetla widgety jeden pod drugim w równych odstępach.
GuiWidget	Reprezentuje podstawowy element gui. Wszystkie elementy gui są widgetami. Widget ma ustaloną geometrię i położenie na ekranie.
GuiWindow	Rozmiar tego widgetu zależy od rozmiaru okna aplikacji. GuiWindow przechowuje jeden widget jako dziecko, który wyświetla.
SettingsAbstract	Abstrakcyjna klasa, która ustala jakie metody powinny implementować klasy typu Settings.
SettingsGame	Reprezentuje ustawienia gry.
SettingsKeybinding	Reprezentuje ustawienia klawiszy.
SettingsMap	Reprezentuje ustawienia mapy gry.
SettingsPlayer	Reprezentuje ustawienia graczy.
BitmaskManager	Przechowywuje tzw. “Maski” wykorzystywane podczas testowania kolizji.
Collision (nagłówek)	Zestaw funkcji do testowania kolizji.
NonCopyable	Klasy dziedziczące NonCopyable mają prywatny konstruktor kopiujący i operator przypisania.
ResourceManager	Pozwala na zapisywanie tekstur, czcionek i dźwięków pod zadaną nazwą i wyciąganie tych obiektów.
SoundRecycler	Usuwa nie wykorzystywane w grze obiekty dźwiękowe.
State	Interfejs stanu gry.
StateMachine	Przechowuje stany gry i je uruchamia na rządanie.
StringUtil (nagłówek)	Funkcje do konwersji liczb na std::string i na odwrót.
Bullet	Interfejs pocisku wykorzystywany w logice gry.
BulletMinigun	Implementacja pocisku typu „minigun bullet”
Collidable	Interfejs dla obiektów, które mogą ze sobą kolidować.
Flag	Reprezentuje flagę w grze.
GameLoop	Pętla gry, która odświeża stan co parę milisekund i aktywuje stany w StateManager.
Ground	Reprezentuje skały w grze. Pozwala na drążenie w nich dziur.
Gun	Interfejs dla działa.
GunMinigun	Implementacja metody shootStrategy(), która zwraca listę pocisków umieszczanych w świecie gry.
Hud	Ma przypisanego gracza i wyświetla na ekranie jego statystyki.
Jetman	Reprezentuje katapultę w świecie gry.
Player	Przechowuje statystyki gracza oraz przypisane do niego działa i samoloty.
Ship	Reprezentuje samolot w grze.
StateGameplay	Wyświetla i obsługuje „grę właściwą”.
StateMenu	Wyświetla menu gry.
Vehicle	Reprezentuje abstrakcyjny samolot w świecie gry.

Inne zastosowania

Jedna z hipotetycznych aplikacji, w której można by wykorzystać jakąś z klas z „AngryJets” to inna gra. SoundRecycler to klasa, która dba o to, żeby nieodtwarzane dźwięki były usuwane. Nie ma problemu, żeby był wykorzystany w innej grze bo nie ma żadnych wzajemnych zależności z obiektami z „AngryJets”.

Kolejny przykład to zbiór funkcji do wykrywania kolizji. Wystarczy, że klasy nowej gry będą implementowały interfejs Collidable i wszystko będzie działać.