Uniwersytet
ŁÓDZKI

Inżynierska praca dyplomowa

Platformowa gra zręcznościowa z wykorzystaniem biblioteki SDL

Autor:

Kamil Lolo

Promotor:

dr.Krzysztof Podlaski

Wydział Fizyki i informatyki stosowanej

1 marca 2013

Spis treści

Wstęp

Głównym celem tej pracy jest stworzenie gry która będzie działać w systemie Linuks, na który to obecnie jest nieporównywalnie mniej gier niż dla komercyjnego systemu Microsoft Windows. Linuks obecnie najczęściej znajduje zastosowanie jako oprogramowanie serwera, superkomputera. Rośnie jednak jego pozycja jako system dla komputerów biurkowych, chodź tutaj nadal uważany jest za system wyłącznie dla tzw. Geeków, czyli maniaków komputerowych z bardzo dużą wiedzą. Pogląd ten stopniowo zmieniany jest przez takie dystrubucje jak Ubuntu. Jest ona równie łatwy w użytkowaniu dla przeciętnego człowieka jak system Windows. Ciągle jednak Linuks nie cieszy się popularnością wśród graczy i twórców gier. Tendencja ta zaczęła się zmieniać w ciągu kilku ostatnich lat, czego dowodem może być wydanie na Linuksa przez firmę Valve Corporation systemu dystrybucji gier Śteam". Jest to niewątpliwie krok do przodu jeżeli chodzi o zmianę poglądu twórców gier że na Linuksa nie warto wydawać gier. Warto tutaj wspomnieć jeszcze o tym że Valve nie jest mało znaną firmą, bardzo wielu graczy kojarzy ją z grą Counter Strike, w którą przez kilka lat grały setki tysięcy ludzi na całym świecie.

Założenie że aplikacja ma działać natywnie w Linuskie. wykluczało użycie narzędzi nie kompatybilnych z tymże systemem, przykładem może być tutaj często wykorzystywany przy tworzeniu gier DirectX firmy Microsoft. Wybór padł natomiast na wieloplatformową bibliotekę Simple Direct Media Layer (w skrócie SDL) której lista docelowych platform jest bardzo długa. Zaczynając od Linuksa, poprzez Windows, Mac OS aż do takich egzotycznych systemów jak Amiga OS. Dodatkową zaletą biblioteki SDL jest fakt że stanowi ona wolne oprogramowanie open source na licencji „zlib". SDL posiada także kilka dodatkowych bibliotek stanowiących rozszerzenie jej możliwości. W aplikacji zostaną wykorzystane dodatkowe moduły rozszerzające API SDL-a o obsługę dźwięku oraz czcionek. Nie są to jednak wszystkie dostępne wtyczki do SDL-a, warto wspomnieć też o bibliotece

SDL_Net dzięki której możliwe jest stworzenie gry sieciowej. Najważniejszymi możliwościami jakimi dysponuje SDL jest utworzenie kontekstu graficznego i obsługa zdarzeń, biblioteka pozwala również za pomocą zestawu funkcji renderować obraz. Rysowanie za pomocą SDL-a często okazuje się jednak zbyt wolne, twórcy biblioteki pozwolili obejść ten problem poprzez wykorzystanie do renderowania niskopoziomowej biblioteki OpenGL, która również jest kompatybilna z Linuksem.

Następnym założonym celem aplikacji było wykorzystywanie zewnętrznych skryptów (tzw. skryptowanie) które dawałyby możliwość manipulowania pewnymi danymi aplikacji bez potrzeby jej re-kompilacji. Skrypty te będą napisane w języku Lua, który został zaimplementowany w ANSI C, dzięki czemu zapewnia wysoką wydajność i przenośność na wiele platform. Ogromnym plusem połączenia programu napisanego w C++ oraz Lua jest to że z poziomu aplikacji C++ można wywoływać funkcje zadeklarowane w skrypcie, a mogą one być zmieniane bez potrzeby rekompilacji aplikacji. Funkcje umieszczone w skrypcie są uruchamianie podczas działania aplikacji przez maszynę wirtualną Lua. Cały ten mechanizm działa również w drugą stronę z poziomu skryptu Lua można wywołać funkcję C++, co też zostanie wykorzystane w aplikacji.

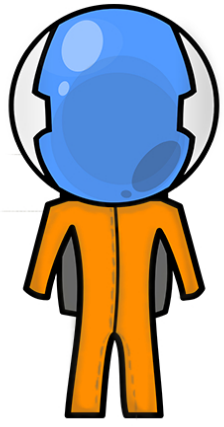
Stworzona na potrzeby pracy gra będzie posiadać grafikę 2D, a dedykowanym systemem operacyjnym będzie linuks, chodź dzięki zastosowaniu wieloplatformowych narzędzi pozostaje możliwość uruchomienia jej w innych systemach np. Microsoft Windows, bądź też Mac OS. Warto tutaj wspomnieć także o mobilnym systemie Blackberry który to wspiera wszystkie technologie które będą użyte w pracy. Daje to możliwość umieszczenia gry w Black Berry App Word – markecie z aplikacjami na urządzenia mobilne z tymże systemem. Co wiąże się z możliwością zarobienia pieniędzy na tej grze. Reasumując. W pracy zostanie przedstawiona aplikacja pokazująca możliwości SDL-a jako biblioteki do tworzenia gier. Przedstawiona zostanie również możliwość wykorzystania skryptów w języku Lua jako narzędzia pozwalającego przenieść część logiki aplikacji poza skompilowany program.

Rozdział 1

Wprowadzenie

1.1 Fabuła

Celem gracza będzie przebiec jak największy dystans. Będzie to bieg astronauty przez obcą planetę na której musi zbierać bańki z tlenem oraz omijać przeszkody żeby nie zginąć. Przeszkodami takimi będą meteoryty zmniejszające poziom życia. Astronauta podczas gry będzie cały czas biec do przodu,zwalniając jedynie w przypadku niskiego poziomu życia.



Rysunek 1.1: Postać astronauty

Poziom życia będzie ciągle spadał, i bańki z tlenem będą go zwiększać. Gracz będzie miał możliwość podskakiwania astronautą oraz wznoszenia się nim do góry Sporadycznie na mapie będą się pokazywać bonusy który astronauta będzie mógł zebrać (maksymalnie 3 naraz) i wykorzystać później do uzupełnienia ilości życia. Taki bonus będzie dawał także nieśmiertelność przez kilka sekund, wtedy to na brzegach ekranu pojawi się charakterystyczna obwódka. Kiedy gracz zakończy grę, wtedy jego wynik, czyli ilość przebytych metrów zapisywany będzie na liście 10 najlepszych

wyników. Wyniki te będą zapisywane w osobnym pliku na dysku, tak żeby dane nie zostały stracone po wyłączeniu aplikacji. Listę najlepszych wyników będzie można obejrzeć wybierając z głównego menu pozycje „highscore". Ze względu na fabułę z biegnącym astronautą, oraz osadzenie zdarzeń na obcej planecie, gra została nazwa „Astro Rush".

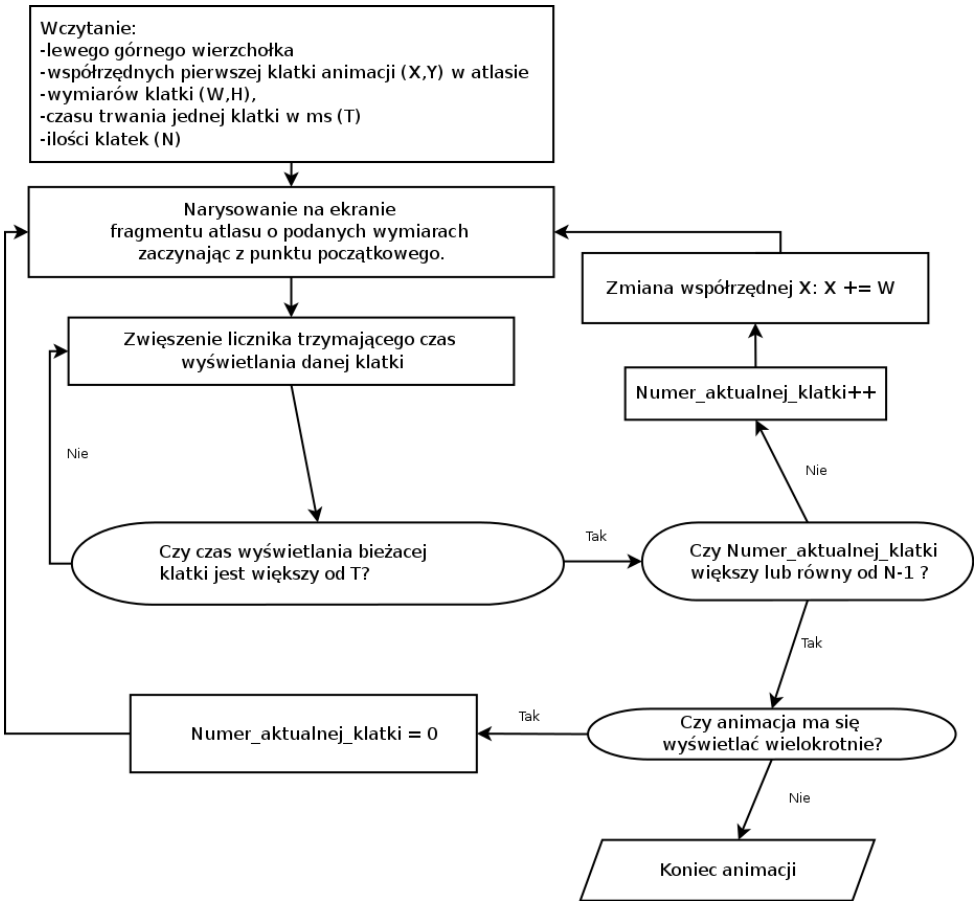
1.2 Grafika

Grafika w grze będzie się opierać o tzw. sprite. Technika ta polega na tworzeniu animacji, bądź też rysowania dużych obrazów z serii małych obrazków które zazwyczaj są fragmentami jednej grafiki. Poszczególne klatki animacji biegu astronauty przedstawia na rysunek 2.

Rysunek 1.2: Animacja biegu astronauty

Wszystkie animacje w grze są przechowywane w jednym głównym pliku „atlas.png". Skąd wyświetlany jest tylko fragment odpowiadający danemu sprite-owi. Po upływie określonego czasu następuje przejście do następnej klatki animacji, czyli zazwyczaj przesuniecie współrzędnej X o szerokość obrazka. W tym algorytmie współrzędna Y nie zmienia się. Cały algorytm wyświetlania animacji opartej przestawia schemat 1. Warto tutaj wspomnieć o układzie współrzędnych jaki jest używany w bibliotece SDL. Otóż punkt początkowy (0,0) znajduje się w lewym górnym rogu, prawy górny wierzchołek to (szerokość okna, 0), natomiast lewy dolny to : (0, wysokość okna). Grafika na potrzeby gry została częściowo stworzona w edytorze grafiki wektorowej Inkscape, który oparty jest na licencji GPL i działa pod takimi systemami operacyjnymi jak np. Windows, Linux. Narysowanie części obrazków jako grafiki wektorowej pozwoliło zachować pełną skalowalność w dalszym procesie tworzenia grafiki. Utworzone grafiki wektorowe były składane i poprawiane w Adobe Photoshop – bardzo rozbudowanej aplikacji do obróbki grafiki rastrowej. Photoshop jest aplikacją płatną, jednak istnieje możliwość użycia 30 dniowej wersji Trial, co też zostało zrobione podczas tworzenia gry. W atlasie grafiki znalazły się

także ikony z kolekcji „Hand drawn icon set” które autor opublikował w internecie na darmowej licencji.



Rysunek 1.3: Schemat wyświetlania animacji opartej o sprite

1.3 Użyte narzędzia

Eclipse

Środowiskiem w którym będzie powstawać aplikacja będzie Eclipse IDE. Znane jest ono przede wszystkim jako bardzo dobre narzędzie do pisania aplikacji w Javie, ale dzięki doinstalowaniu wtyczki CDT (C/C++ Development tooling) można w nim rozwijać projekt w języku C++. Eclipse posiada integracje z wieloma przydatnymi narzędziami, które czasami bardzo upraszczają życie programiście. Przy tworzeniu Astro Rush były wykorzystane: -Integracja Eclipse z debuggerem. W tym przypadku znany konsolowy gdb. Najbardziej przydatna okazała się tutaj możliwość zatrzymania programu na breakpoincie oraz sprawdzenie stosu wywołań. -Integracja z make, co zostanie omówione w kolejnym

podrozdziale -Możliwość doinstalowania kolejnych wtyczek. Przy tworzeniu przydatna okazała się wtyczka aplikacji Valgrind jako narzędzie do wykrywania wycieków pamięci, oraz integracja z aplikacją Perf. Jest to profiler który pokazuje statystyki odnośnie tego które wywołania metod trwają najdłużej. Profilowanie okazało się pomocne podczas refactoringu w ramach którego została przeprowadzona optymalizacja wydajności.

Make

Eclipse jako platforma programistyczna dedykowany jest językowi Java, a kolejne wtyczki pozwalające pisać w innych językach to tylko rozszerzenie tego środowiska Javy. Eclipse z wtyczką do języka C++ może używać programu Make, który automatyzuje budowanie projektu składającego się z wielu plików. Zostało to wykorzystane w projekcie. Ogromnym plusem takiego rozwiązania jest to żeby zbudować aplikacje nie potrzeba ściągać Eclipse-a, ale wystarczy make, który zajmuje niewiele miejsca na dysku i jest wieloplatformowy. Dodatkowo bardzo łatwo się go instaluje w większości dystrybucji Linuksa. Make korzysta z pliku reguł domyślnie o nazwie „makefile". Taki plik dla omawianej gry wygląda następująco:

```
CXX = g++
CFLAGS = -Wall -ansi -pedantic -g -std=c++0x -Wall -I ./include -O0 -c

# flagi linkera
LIBS = -lGL -lGLU -lSDL -lSDL_mixer -lSDL_ttf -lSDL_image -lluabind -llua5.1

# lista plikow źródłowych do kompilacji
SOURCES = src/main.cpp src/App.cpp src/Property.cpp src/Resource.cpp

# jak maja się nazywać skompilowanie pliki cpp
OBJECTS=$(SOURCES:.cpp=.o)

# nazwa pliku wynikowego
EXECUTABLE = AstroRush.bin

# domyslny cel dla wywolania make bez argumentu, czyli zbudowanie projektu
all: $(SOURCES) $(EXECUTABLE)
```

```
# linkowanie aplikacji
$(EXECUTABLE): $(OBJECTS)
 @echo "\n ---- Linkowanie ---- "
$(CC) $(OBJECTS) -o $(EXECUTABLE) $(LIBS)

#kompilowanie plikow cpp
.cpp.o:
 @$(CXX) $(CFLAGS) $< -o $@

# czyszczenie aplikacji przed zbudowaniem
clean:
rm -rf ./src/*.o
rm ./AstroRush.bin
```

Domyślne wywołanie make bez żadnych parametrów spowoduje zawsze uruchomienie domyślnego celu budowania: all. Możliwe jest definiowania dowolnej ilości celów budowania w jednym makefile-u. W grze został dodany również cel do czyszczenia gry z wszystkich skompilowanych źródeł oraz zlinkowanej aplikacji, co okazuje się przydatne kiedy występuje potrzeba przebudowania projektu, bowiem make sprawdza czasy ostatniej modyfikacji plików tzw. timestampy i kompiluje tylko te źródła które uległy zmianie od ostatniej kompilacji. W napisanym na potrzeby gry pliku „makefile" widać że w plikach makefile można definiować swoje własne zmienne, tutaj na przykład zmienną są flagi linkera, kompilatora, oraz lista plików źródłowych z katalogu src. Tak napisany makefile jest bardzo uniwersalny i kolejne nowe pliki wymagają jedynie dopisania ich na listę źródeł. Ewentualnie podczas kompilacji w systemie w którym w zmiennej środowiskowej nie ma kompilatora g++ można tutaj podać ścieżkę gdzie ten kompilator się znajduje. Budowanie aplikacji poprzez make bądź też podobne narzędzie o nazwie cmake jest wyjątkowo popularne w systemie Linuks i projektach napisanych w języku C/C++. Make jest nawet wykorzystywany do budowania jądra Linuksa, które składa się z milionów linii kodu (wersja 3.2 to w przybliżeniu 15 mln) oraz setek plików, gdzie nie wszystkie muszą być skompilowane, a przy rekompilacji kompilowane są tylko te które uległy zmianie, dzięki czemu oszczędność czasu przy kompilacji jest znacząca.

Git

Projekt Astro Rush nie wydaje się zbyt duży biorąc pod uwagę fakt że nie przekroczył 10 tysięcy linii kodu. Jednak zawsze warto mieć jakąś kopie na repozytorium oraz ewentualnie możliwość poprzez historie zmian przywrócenie jakiś fragmentów kodu. Narzędziem które okazało się tutaj pomocne jest rozproszony system kontroli wersji – git. Darmowe oprogramowanie stworzone przez Linusa Torvaldsa do zarządzania kodem jądra Linuksa. Git jest również narzędziem wieloplatformowym, chodź pod systemem Windows jest on wolniejszy niż na Linuksie. Przy tworzeniu aplikacji został wykorzystany serwis hostujący gita: https://github.com/. Hosting dla aplikacji open source jest darmowy, jednak w takim przypadku repozytorium z kodem jest publiczne. Warto wspomnieć o tym że serwis github mimo tego że powstał dość nie dawno bo w 2008, ma już 2 miliony repozytoriów.

Git w środowisku Linuks jest narzędziem konsolowym, jednak w ramach ułatwienia podczas tworzenia aplikacji została wykorzystana wtyczka do eclipse która pozwala w łatwy sposób synchronizować projekt który znajduje się na dysku lokalnie z tym co jest na repozytorium, oraz wysyłanie zmian na serwer.

1.4 OpenGL jako silnik grafiki

Czym jest OpenGL

Open Graphics Library (w skrócie OpenGL) jest niskopoziomową biblioteką graficzną 3D. Kompatybilny jest on z większością liczących się systemów operacyjnych, został on także zaimplemtowany na urządzeniach mobilnych, przykładem może być tutaj JOGL, czyli Java-owa wersja OpenGL-a której można używać na urządzeniach z systemem Android. OpenGL jest często wykorzystywany jako podstawowe API przy tworzeniu silników do gier 3D przykładem może być tutaj choćby nawet silnik ID tech znany min. z serii gier Quake . Mimo że OpenGL jest przystosowany do pracy z grafiką trójwymiarową to doskonale można go wykorzystać do grafiki 2D, tak jak to miało miejsce w grze Astro Rush. OpenGL posłużył do wyświetlania tekstur na ekranie, co odbywało się zdecydowanie szybciej niż poprzez funkcje do rysowania z biblioteki SDL. Różnica w szybkości renderowania wynosiła około 20 fps-ów na sekundę na laptopie hp550 z procesorem dual core 1.4 ghz.

OpenGL dostarcza także takich zaawansowanych elementów jak obsługa cieni oraz oświetlenia, jednak z racji wykorzystania grafiki 2D nie znalazło to zastosowania w projekcie. W bardzo łatwy sposób można połączyć SDL-a i OpenGL-a. W SDL podstawowym elementem graficznym na którym odbywa się rysowanie jest powierzchnia (ang. surface). Podczas inicjowania biblioteki SDL tworzona jest główna powierzchnia na której następnie będzie się odbywać rysowanie (często też nazywane w grafice 2D „blitowaniem"). Podczas inicjowania głównej powierzchni ekranu żeby używać do renderowania OpenGL-a wystarczy poprzez funkcje SDL_SetVideoMode podać flagę SDL_OPENGL. Trzeba jeszcze pamiętać żeby po każdym rysowaniu wywołać funkcję: SDL_GL_SwapBuffers() która wysyła bufor ramki do rysowania na ekranie.

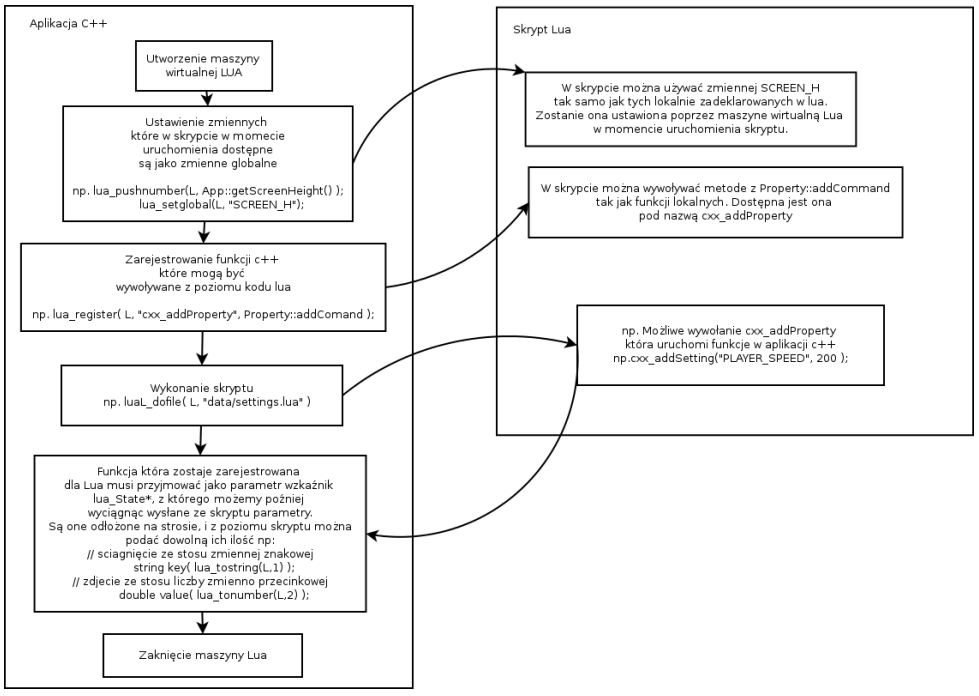
Wykorzystanie OpenGL do renderowania grafiki w grze

SDL posiada rozszerzenie SDL_image które umożliwia obsługę różnych formatów grafiki (w grze zostały wykorzystane pliki graficzne o rozszerzeniach png oraz jpeg). Pozwala one w łatwy sposób wczytać plik z dysku poprzez funkcje SDL_Surface *IMG_Load(const char *file) , która zwraca surface z wczytanym obrazkiem. SDL_Surface jest strukturą w której znajduje się wskaźnik do pamięci gdzie znajduje się wczytana z dysku grafika. Ten adres („void* pixels") należy przekazać jedynie do OpenGL-a podczas tworzenia tekstury. Można w ten sposób rysować nie tylko pliki graficzne wczytane z dysku ale również obiekty graficzne utworzone w aplikacji. Taka sytuacja ma miejsce w przypadku renderowania napisów, gdzie jest tworzony surface z napisem, i następnie rysowany za pomocą OpenGL-a. W aplikacji proste prymitywy graficzne jak prostokąty wypełnione kolorem są rysowane również za pomocą API OpenGL-a.

1.5 Skrypty w języku Lua

Lua jest lekkim językiem skryptowym zaprojektowanym do rozszerzania możliwości innych aplikacji. Został on zaimplementowany w języku C zgodnie ze standardem ANSI, zapewnia mu to przenośność na wiele platform. Najważniejszym cechami tego języka jest to że jest dynamicznie typowany, oraz obiektowy. Jest on wykorzystywany zarówno do tworzenia rozszerzeń do różnych aplikacji, co często nazywane jest skryptowaniem

(ang.scripting) oraz jako samodzielny język, w którym skrypty będą wykonywane poprzez maszynę wirtualną Lua. Samo skryptowanie wiąże się z ideą programowania sterowanego danymi (ang. data driven development). Podejście te zakłada że wszelkie stałe kontrolujące zachowanie programu oraz wybrane elementy logiki powinny być zdefiniowane poza program. W aplikacji skrypty Lua są wykorzystywane do przechowywanie wszystkich ustawień, oraz obliczania pewnych wartości już w czasie działania aplikacji.Przykładowo rozmiar gracza jest obliczany na poziomie skryptu przy wykorzystaniu ustawionych podczas działania gry zmiennych z rozmiarami ekranu.



Rysunek 1.4: Przepływ danych między aplikacją C++ a skryptem Lua

Powyższy rysunek pokazuje przykładowy przepływ danych między aplikacją C++ a skryptem Lua. Wykorzystane tu zostało wywołanie funkcji C++ z poziomu skryptu, aczkolwiek w drugą stronę ten mechanizm również działa. To znaczy z poziomu C++ można wykonać funkcje Lua. Lua została wykorzystana do internacjonalizacji aplikacji. Podobnie jak to jest wykorzystywane w aplikacjach webowych w języku Java, gdzie wszystkie komunikaty są przechowywane w plikach properties. Podczas uruchamiania aplikacji zostaje odczytany odpowiedni plik dla danego języka. Cały proces ilustruje rysunek 5. W rezultacie takiej budowy aplikacja może działać z dowolnym językiem. Podczas tworzenia zostały napisane komunikaty zarówno polskie jak i angielskie.

1.6 Wprowadzenie do biblioteki SDL

Rozdział ten będzie zawierał wprowadzenie do tworzenia aplikacji z wykorzystaniem biblioteki SDL, nie będzie tu poruszony temat instalacji tej biblioteki. Zagadnienie te będzie znajdować się w rozdziale dotyczącym kompilacji projektu w systemie linuks.

Simple Direct Media Layer jest biblioteką ułatwiającą tworzenie gier komputerowych, oraz różnych aplikacji multimedialnych. Umożliwia ona stworzenie okna, oraz zarządzanie nim. Dodatkowo zapewnia obsługę zdarzeń związanych z klawiaturą, myszą oraz joystickiem. Możliwa jest nawet obsługa CD-ROM-u za pomocą tego API. Największą zaletą obok dużej funkjonalności jest prostota aplikacji pisanej z wykorzystaniem tej biblioteki. Najprostszy program może wyglądać następująo:

Tak napisany program można skompilować za pomocą kompilatora gcc z poziomu linuksowej konsoli za pomocą polecenia:

Aplikacja po uruchomieniu utrzowy okno które zostanie natychmiast zamknięte. Takie działanie aplikacji jest mało przydatnego, dlatego też konieczna jest pętla w której będzie odbywać się praca całego programu. Program wykorzystujący taką pętle może wyglądać nastepująco: kod kod kod

(Opis oblugi zdarzen)

SDL dostarcza również obsługę wielowątkowości oraz timerów. Dzięki czemu nie potrzebna była w projekcie żadna dodatkowa biblioteka która umożliwiała by utworzenie timera. (opis timerow)

1.7 Obsługa czcionek w SDL

W projekcie została wykorzystana biblioteka SDL_ttf pozwalająca używać w aplikacji czcionek w formacie True Type. Format ten stworzony przez firmę Apple przechowuje kształty poszczególnych liter jako krzywe Beziera, i jest on obsługiwany przez większość platform. Na Linuksie jest on bardzo powszechnym formatem do obsługi czcionek, dodatkowym plusem jest ogromna ilość czcionek na darmowych licencjach. W grze została wykorzystana czcionka "Ubuntu"udostępniona za darmo, i będąca domyślną czcionką w

dystrybucji Linuksa o tej samem nazwie. Plik z taką czcionką (standardowo o rozszerzeniu *.ttf) jest wczytywany podczas uruchamiania aplikacji, następnie poprzez wywołania funkcji z biblioteki SDL_ttf np. TTF_RenderUTF_Blended zostaje utworzona powierzchnia na której narysowany jest napis o podanej treści, kolorze oraz rozmiarze.

Niestety powierzchnia taka jest zwracana jako wskaźnik na strukturę SDL_Surface, przez co konieczna jest konwersja na format obsługiwany przez OpenGL-a. Niedogodność taka nie występowałaby gdyby do renderowania było wykorzystywane API biblioteki SDL. Identyczny problem występuje również przy renderowaniu innych elementów graficznych które są ładowane z dysku i zwracane jako SDL_Surface* (Wczytywanie takie realizowane jest poprzez kolejną bibliotekę będącą uzupełnieniem SDL-a: SDL_image. Służy ona do wczytywania plików graficznych w takich formatach jak np. JPEG, PNG, TIFF. W aplikacji wykorzystana jest tylko jedna funkcja z tej biblioteki stąd też nie będzie ona szerzej omawiana).

Sama konwersja SDL_Surface* na GLuint to wygenerowanie tekstury w standardowy dla OpenGL-a sposób, wykorzystując przy tym pole pixels ze struktury SDL_Surface, które jest adresem pod którym przechowywane są poszczególne piksele obrazka. W uproszczeniu funkcja realizująca taką konwersje w grze wygląda następująco:

```
void RendererGL::create_gl(SDL_Surface * surf, GLuint * tex )
{

   /** ...tutaj określenie ilości kolorów i formatu */

    glGenTextures( 1, tex );
    glBindTexture( GL_TEXTURE_2D, *tex );

    /** ...tutaj ustawienia parametrów tekstury */

    glTexImage2D( GL_TEXTURE_2D, 0, colors_amount,
        surf->w, surf->h, 0, format,
        GL_UNSIGNED_BYTE, surf->pixels );
}
```

Warto wspomnieć że biblioteka SDL_ttf pozwala renderować napisy z polskimi znakami, o ile takie występują w wczytanej czcionce. Ponadto SDL_ttf dostępny jest podobnie jak

SDL na darmowej licencji zlib, i jest wieloplatformowy jak wszystkie wtyczki do SDL-a. W niektórych dystrybucjach zainstalowanie tej biblioteki, oraz innych wspomnianych bibliotek rozszerzających SDL-a sprowadza się do wykonania jednego polecenia- zainstalowania pakietu z repozytorium. Dla dystrybucji Debian oraz jego pochodnych będzie to polecenie:

```
apt-get install libsdl-ttf2.0-dev
```

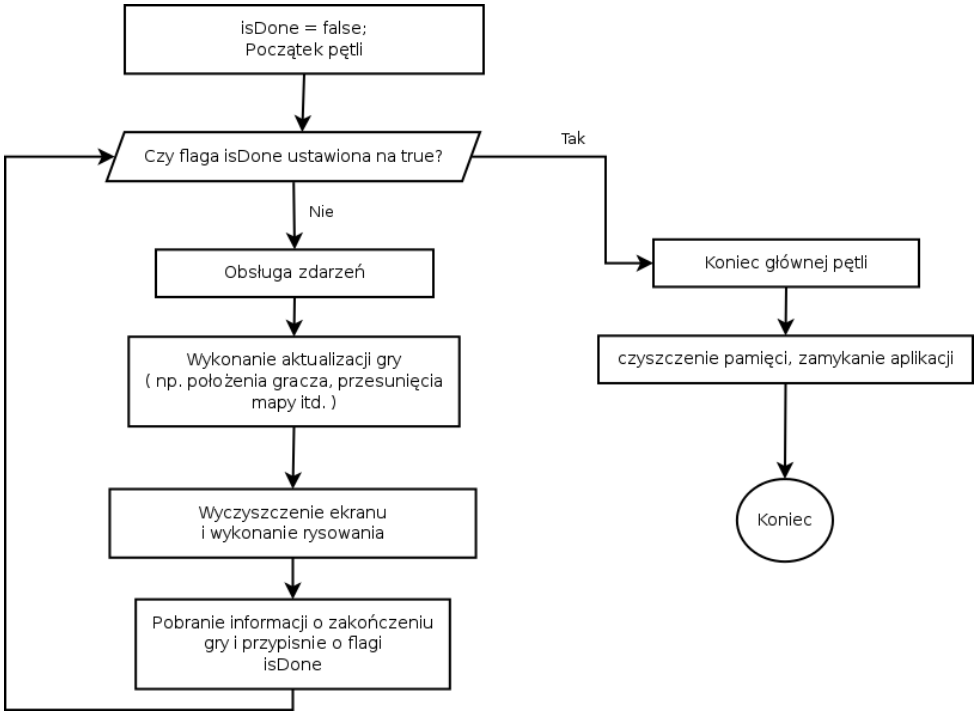
Rozdział 2

Budowa aplikacji

2.1 Główna pętla

W grach komputerowych często wykorzystywane jest tzw. programowanie sterowane zdarzeniami. Polega ono na umieszczeniu w aplikacji głównej pętli, w której to cyklicznie będzie się odbywać obsługa zdarzeń (np. naciśniecie klawisza), aktualizacja gry oraz rysowanie. Sama kolejność tych elementów nie odgrywa większej roli, warto natomiast zwrócić uwagę na to że po zatrzymaniu pętli następuje przygotowanie aplikacji do wyłączenia. W przypadku Atsro Rush po wyjściu z głównej pętli zatrzymywany jest kontekst graficzny SDL-a, zwalniane są wszystkie zajęte zasoby, i następuje wyłączenie gry. Pętla taka najczęściej implementowana jest jak while, którego zakończeniem steruje flaga wyjścia. Przy każdym obiegu pętli wartość tej flagi wyciągana jest z klasy Game, która to decyduje kiedy należy zakończyć działanie aplikacji.

Pętla stanowi najważniejszy element większości gier, to od niej zależy czy gra będzie działać tak samo na urządzeniach różniących się wydajnością. W projekcie pętla jest dość prosta i oprócz typowych elementów typu aktualizacja, rysowanie, obsługa zdarzeń, uwzględnia jedynie sytuacje w której całość obliczeń i rysowań odbywa się zbyt szybko i należy wykonać opóźnienie. Taki problem może się pojawić na szybszych urządzeniach na których gra działała by zbyt szybko. W przypadku bardziej złożonej aplikacji można także uwzględnić sytuacje odwrotną, kiedy to ostatnia aktualizacja stanu gry odbywała się zbyt wolno i w następnym obiegu pętli należy wykonać aktualizacje kilkukrotnie żeby



Rysunek 2.1: Schemat działania głównej pętli

zapobiec braku płynności w renderowanym obrazie. Taka sytuacja jednak w grze Astro Rush nie powinna mieć miejsca z racji tego iż jest jest to gra z grafiką dwuwymiarowa i występują w niej proste obliczenia matematyczne.

2.2 Mapa w grze

Mapa kafelkowa

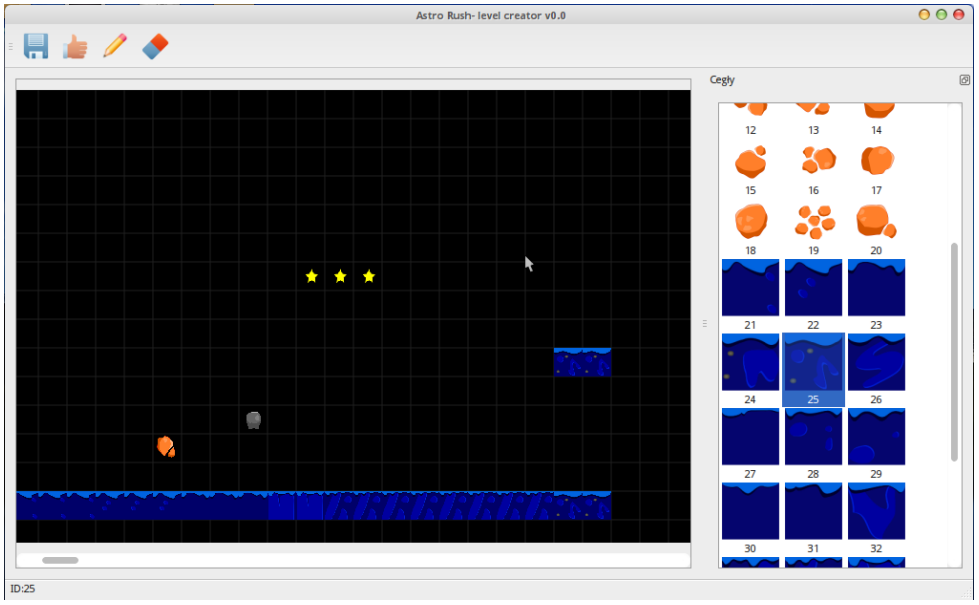
Mapa kafelkowa (ang. tiled map) jest jedną z podstawowych technik przy tworzeniu gier z grafiką dwuwymiarową. Technika ta polega na podziale świata dostępnego w grze na fragmenty (tzw. kafelki) o tych samych rozmiarach. Najczęściej są to kwadraty, którym przypisujemy odpowiednie identyfikatory grafik. Tak utworzona mapa przechowywana jest w postaci dwuwymiarowej macierzy w osobnym pliku na dysku, i jest wczytywana podczas startu aplikacji w osobnym wątku. Macierz składająca się wyłącznie z cyfr (typu short żeby dodatkowo oszczędzić pamięć) zajmuje o wiele mniej pamięci w przeciwieństwie do rozwiązania w którym z dysku wczytywana jest cała mapa w postaci jednej grafiki. Na podstawie tej macierzy rysowana jest mapa widoczna na ekranie.

Z racji tego że gracz ciągle wędruje prze mapę, ta cały czas jest przesuwana, a dokładniej to inkrementowany jest indeks kolumny w macierzy kafelków od której zaczynamy rysowanie. Kolumna o takim indeksie rysowana jest na ekranie jako pierwsza z lewej strony, następnie rysowane są obok (po prawej stronie) kolejne kolumny aż do momentu w którym mapa pokrywa cały ekran. Kolumna od której zaczyna się rysować od lewego brzegu ekranu przesunięta jest w lewo o pewien offset, który zwiększnay jest podczas biegu gracza do przodu. Offset ten sprawia że współrzędna na osi X skrajnej kolumny zostaje przesunięta w lewo po za ekran, tak że widoczny jest tylko fragment kolumny na ekranie. Przejście do następnej kolumny następuje w momencie kiedy skrajna kolumna znajduje się całkiem po za ekranem. Dzięki zastosowaniu takiego algorytmu nastepuje płynne przesuwanie mapy, bez widocznych przeskoków pomiędzy kolejnymi kolumnami.

Edytor mapy

Opisana w poprzednim podrozdziale macierz kafelków w grze Astro Rush ma wymiary: 30000 x 15. Stąd też pojawił się problem edycji tak dużej ilości danych. Zmiana poszczególnych wpisów ręcznie nie wchodziła w grę, dlatego też powstała dodatkowa aplikacja do edycji mapy. W wizualny sposób, z wykorzystaniem jedynie myszy można w niej stworzyć w kilkanaście minut całą mapę, rozmieszczając na niej dostępne rodzaje kafelków. Edytor umożliwia także wczytanie stworzonej wcześniej mapy i jej edycje. Aplikacja została napisana z wykorzystaniem biblioteki Qt udostępnionej na licencji LGPL. Biblioteka ta jest zestawem przenośnych narzędzi do tworzenia między innymi interfejsu użytkownika, obsługi sieci, grafiki trójwymiarowej (OpenGL), plików i wielu innych.

Edytor mapy wyświetla całą planszę w postaci siatki na której naniesione są kafelki. Poprzez kliknięci w daną komórkę możemy zmienić rodzaj kafelka który w danym miejscu ma się wyświetlić, bądź też wyczyścić daną komórkę. Do pliku zapisywane są tylko numery odpowiadającym poszczególnym kafelkom, na bazie których gra rozpoznaje jaką grafikę w danym miejscu wstawić. Numeracja kafelków rozpoczyna się od 0, natomiast wartość -1 oznacza że w danym miejscu nie ma kafelka i taki fragment nie jest rysowany. Edytor jest aplikacją bardzo prostą, wszelkie jego modyfikacje np. dodanie nowego rodzaju kafelka wymaga ręcznych zmian w kodzie programu, jednak na potrzeby pracy takie rozwiązanie okazało się wystarczające.

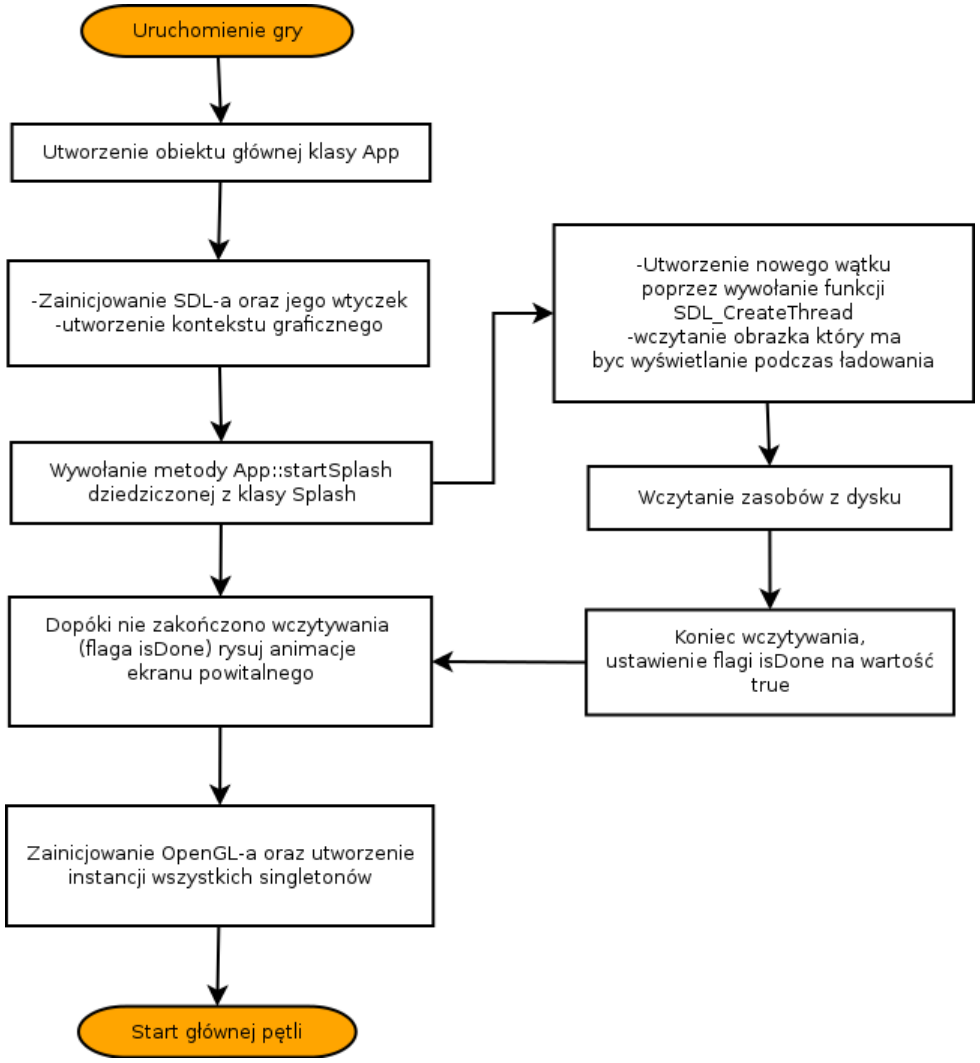


Rysunek 2.2: Edytor mapy podczas pracy

2.3 Uruchamianie aplikacji

Aplikacje takie jak gry wymagają do pracy zewnętrznych zasobów które są przechowywane na dysku. Mogą to być grafiki, dźwięki, czcionki. Wraz ze wzrostem ilości materiałów jakie muszą być załadowane do aplikacji przy jej starcie rośnie też czas oczekiwania gracza na to aż aplikacja będzie gotowa do pracy. Dlatego też wczytywanie zasobów, oraz inne czynności które trwają długo są wykonywane podczas uruchamiania, a użytkownikowi prezentowany jest w tym czasie ekran powitalny (ang. Splash screnn). W grze Astro Rush taki ekran również jest prezentowany. Pokazuje się na nim również pasek postępu obrazujący ile czasu pozostało jeszcze do uruchomienia właściwej części aplikacji. Żeby samo rysowanie ekranu powitalnego odbywało się płynnie wczytywanie danych z dysku odbywa się w osobnym wątku.

Wielowątkowość w projekcie jest możliwa dzięki wykorzystaniu specjalnej funkcji z biblioteki SDL. Funkcja SDL_CreateThread przyjmująca jako argument wskaźnik do funkcji tworzy nowy wątek w którym zostaje uruchomiona ta funkcja. W funkcji takiej zostało umieszczone wczytywanie wszystkich zasobów z dysku. W momencie kiedy funkcja wczyta wszystkie dane, wtedy ustawi flagę oznaczającą zakończenie ładowania na wartość True. Zmiana wartości tej flagi spowoduje zakończenie wyświetlania ekranu powitalnego i uruchomi główną pętle gry. Cała funkcjonalność związana z ekranem powitalnym została umieszczona w osobnej klasie Splash która jest dziedziczona prze główną klasę aplikacji -



Rysunek 2.3: Schemat uruchamiania gry

App. Dzięki takiej implementacji klasa App zajmuję się jedynie inicjowaniem bibliotek, oraz obsługą głównej pętli.

Należy wspomnieć ile miejsca zajmują omawiane zasoby na dysku. W przypadku dźwięków jest to około 10 mb, grafika to 3 mb, a sam plik z mapą to 1.3 mb. Na wczytanie takich ilości danych w zależności od sprzętu może być potrzebne nawet do kilku sekund.

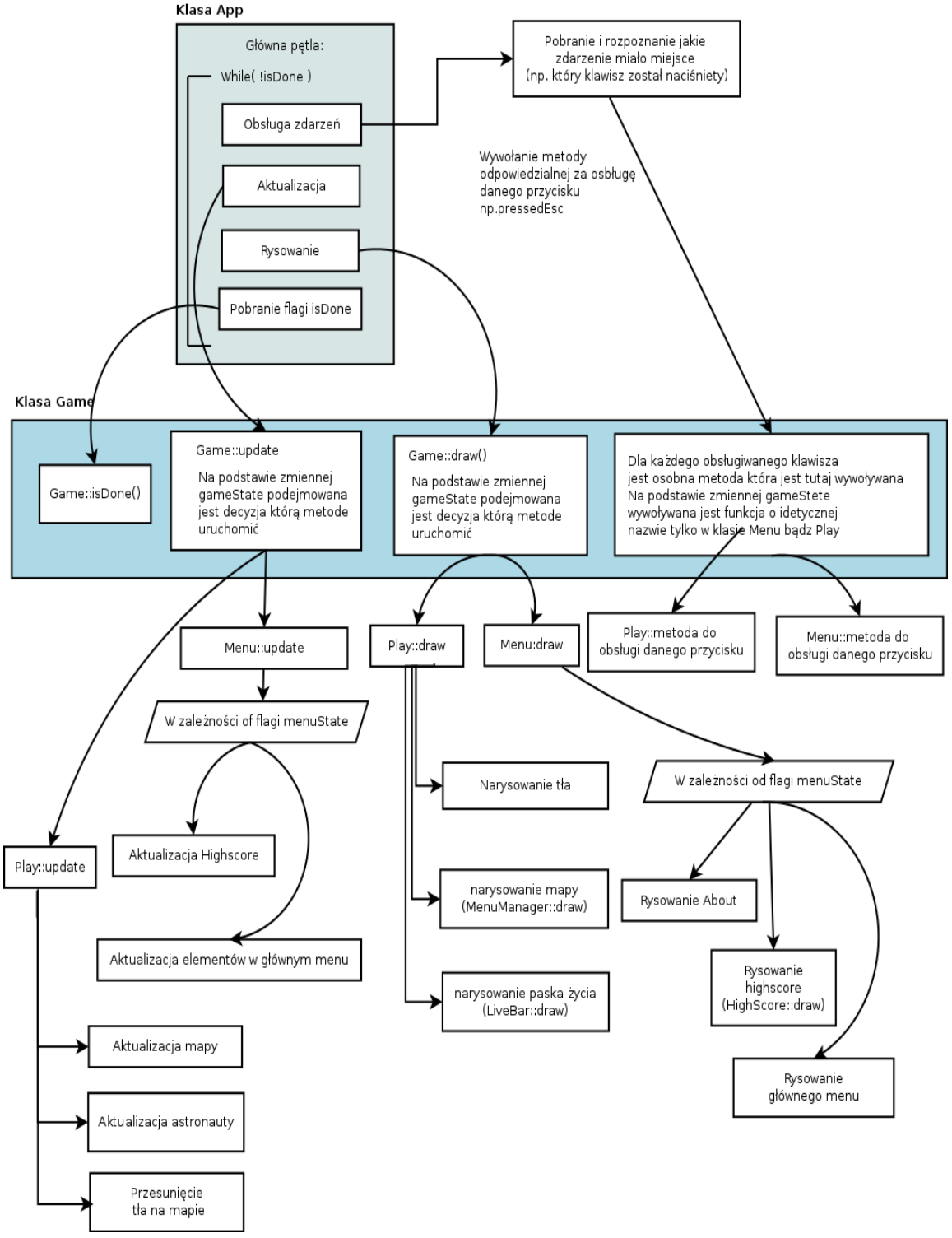
2.4 Przyjęte konwencje, logowanie i obsługa błedów

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut

magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi, imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.

2.5 Warstwy aplikacji

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi, imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.



Rysunek 2.4: Schemat przepływu danych w aplikacji

Rozdział 3

Dokumentacja użytkownika

3.1 Sterowanie w grze

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi, imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.

3.2 Menu

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi,

imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.

3.3 Instalacja

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi, imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.

```
if( int x =0 ) {
        cout<<dupa;
}
```

Zakończenie

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu massa ante. Maecenas pretium metus a libero commodo convallis. Mauris a dignissim lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Duis eleifend magna ut magna commodo dapibus. In adipiscing enim eget sapien elementum et adipiscing ligula sagittis. Curabitur ullamcorper cursus vulputate. Donec dignissim, tortor eget adipiscing rhoncus, risus mauris varius nisi, ac vehicula elit orci sit amet nunc. Fusce massa nisi, imperdiet vitae volutpat non, euismod ullamcorper lectus. Mauris iaculis sagittis tortor, quis convallis elit luctus eu. Sed sodales viverra velit, quis porttitor ipsum vulputate nec.

Etiam imperdiet, elit sit amet vulputate rutrum, metus leo ultrices nibh, quis tempor tortor neque vitae ante. Proin nec nisl id ante consectetur suscipit. Sed ornare aliquet nunc nec congue. Pellentesque laoreet neque nec justo porttitor non condimentum purus mollis. Sed pellentesque rhoncus tortor. Suspendisse a magna quis tellus euismod laoreet vitae at purus. Integer et odio ac nunc lacinia blandit. Suspendisse dignissim vehicula porta. Cras blandit tellus a ante ultricies at viverra ipsum scelerisque. Vivamus at erat sed ipsum consequat elementum eget non tellus.

Nullam justo ligula, pellentesque vel auctor vel, tincidunt ac lacus. Nullam eu magna magna. Suspendisse lacinia dictum suscipit. Pellentesque sed turpis non neque semper vulputate sollicitudin id lectus. Suspendisse vulputate commodo neque, et elementum erat consectetur ac. In hac habitasse platea dictumst. Phasellus quis tellus ipsum. Nam mollis faucibus eros nec faucibus. Ut aliquam interdum dui, sit amet hendrerit neque auctor sit amet.

Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Proin iaculis, libero et ornare condimentum, augue dui pretium justo, ut vulputate justo

leo ut enim. Aliquam condimentum augue rutrum sapien varius luctus. Nulla placerat molestie nunc, nec molestie turpis fringilla in. Aliquam vel massa vitae diam scelerisque bibendum. Fusce fermentum, justo malesuada hendrerit gravida, mi augue ultricies erat, sit amet tristique diam turpis non metus. Etiam volutpat faucibus eros quis fermentum.

Nunc quis lacus velit, at convallis ante. Sed mi arcu, commodo vitae hendrerit sit amet, semper ac mi. Integer tincidunt suscipit dui ac facilisis. Donec eleifend viverra neque, ut lobortis libero molestie id. Suspendisse eu commodo leo. Nam nec ligula at velit fermentum fringilla non eu lectus. Maecenas condimentum porttitor sodales. Duis eget libero ac sapien malesuada convallis vel sed mauris. In hac habitasse platea dictumst. Suspendisse potenti. Fusce pretium blandit sapien, non tempor erat commodo at. Vestibulum interdum consequat nunc sed faucibus. Vestibulum in dui sit amet nulla accumsan dapibus eu nec felis. Nunc et augue eros, at aliquam quam. Vestibulum sapien mauris, feugiat a convallis vitae, volutpat vitae augue. Vestibulum aliquam laoreet metus ac aliquet.

Spis rysunków

Bibliografia

[1] Janusz Ganczarski. *OpenGL w praktyce*. Wydawnictwo BTC, 2008.

[2] Ernest Pazera. *Focus o SDL*. Premier Press, 2003