Systemy multimedialne

Projekt: Air Hockey

Skład sekcji 4:

Grzegorz Kokoszka
Jakub Mendel
Piotr Okoń
Kamil Wojciechowski

Data opracowania:

12.06.2017

Spis treści

Cel projektu

Celem projektu było utworzenie adaptacji gry Air Hockey napisanej w języku **C++** w środowisku **2d**, której główną cechą jest sterowanie przy użyciu kamerki internetowej. Miały być zrealizowane trzy tryby gry:

- tryb jednoosobowy z przeciwnikiem sterowanym przez komputer
- tryb dla dwóch graczy przy jednym komputerze
- tryb gry w sieci lokalnej

Sterowanie miało się odbywać przez detekcję jednokolorowych okrągłych wskaźników przy częstotliwości rejestracji klatek pozwalającej na komfortową obsługę. Warstwa graficzna oraz komunikacja sieciowa miała być zrealizowana przez bibliotekę **SFML**, która jest prosta ale spełniająca nasze wymagania. Komunikacja z kamerką internetową oraz narzędzia do przetwarzania obrazu w czasie rzeczywistym miała być zrealizowana dzięki potężnej bibliotece **OpenCV**. Pobocznym celem było napisanie prostego interfejsu graficznego dostosowanego do naszych potrzeb z uwagi na nie stosowanie dodatkowych bibliotek rozszerzających **SFML** np. **SFGUI**. Dodatkowym celem było napisanie aplikacji działającej na platformie **Windows** oraz **Linux** co ułatwiły wybrane przez nas biblioteki.

Dokumentacja użytkownika

Wymagania sprzętowe

- Karta graficzna z obsługą OpenGL >= 2.0
- Kamerka internetowa z możliwością nastawy ekspozycji oraz FPS >= 25
- Pamięć RAM >= 1 GB
- Rozmiar aplikacji ok 10 MB
- Procesor minimum 2 rdzenie np. Intel® i3-530 2.93 GHz

Wymagania fizyczne (wskaźniki)

Do prawidłowego działania aplikacji potrzebne są jednokolorowe wskaźniki. Kształt zarejestrowanego wskaźnika powinien być okrągły.

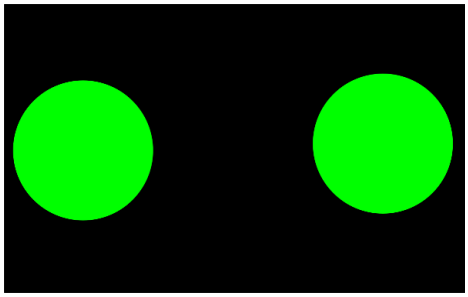
Proponowane przez nas wskaźniki to piłeczki pingpongowe z wyciętą dziurą do zakładania na palec (wskazujący oraz kciuk).

Rysunek 1: Piłeczka pingpongowa

Rysunek 2: Wyścielane piłeczki

Dla lepszej wygody i dopasowania można wyścielić dziurę tkaniną, przymocować ją np klejem na gorąco.

Należy pamiętać aby kolor wskaźników był unikatowy w obrębie sceny, co pomoże uniknąć niemożliwych do odfiltrowania szumów tła. Wskaźniki białe również są możliwe do zastosowania lecz wymagają większych starań odnośnie tła (odbicia światła słonecznego i sztuczne źródła światła mogą być źródłem niepożądanych zakłóceń).

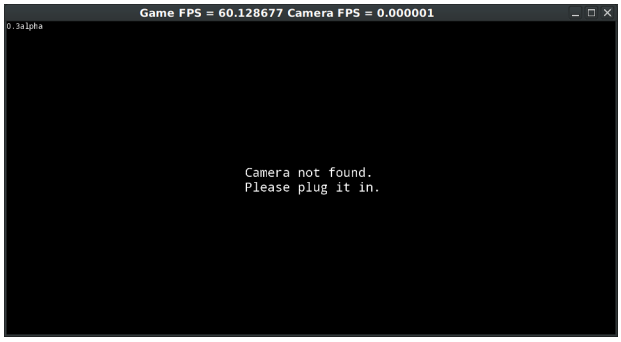


Rysunek 3: Alternatywny wskaźnik z obrazka na smartfonie

Instrukcja obsługi

Uruchomienie aplikacji

Wykonywalny plik aplikacji to **airhockey.exe** (Windows) lub **airhockey** (Linux). Po jego otworzeniu ukaże się napis:



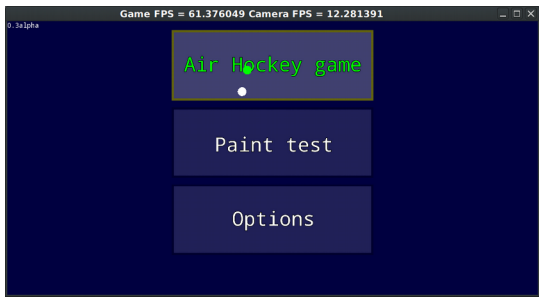
Rysunek 4: Informacja o konieczności podłączenia kamerki

Jeżeli mamy podłączoną kamerkę to napis powinien zniknąć w ciągu kilku sekund. Jeżeli mimo podłączenia nadal występuje należy dowiedzieć się pod jakim identyfikatorem występuje i spróbować ustawić go w pliku program.conf.

Menu główne aplikacji

Główne menu pozwala przejść do:

- Podmenu gry Air Hockey
- Paint testu
- Ustawień

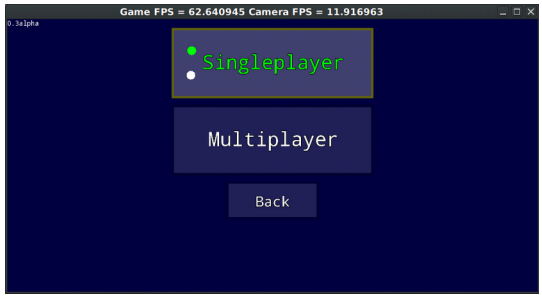


Rysunek 5: Menu główne

Menu gry Air Hockey

W menu gry Air Hockey mamy do wyboru:

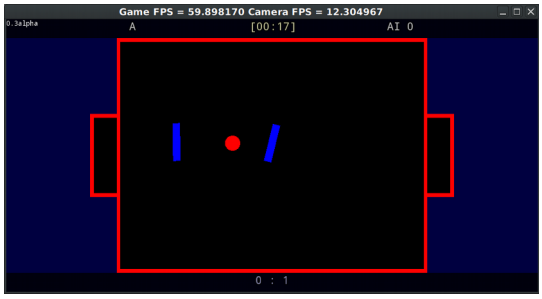
- Grę jednoosobową z przeciwnikiem sterowanym przez komputer.
- Grę wieloosobową
- Powrót do menu głównego



Rysunek 6: Podmenu wyboru gry

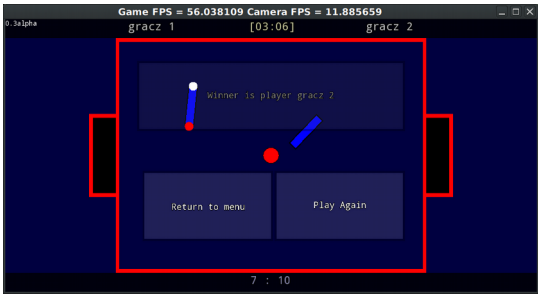
Przebieg gry

Na załączonym obrazku przedstawiony jest widok gry w trybie jednoosobowym.



Rysunek 7: Widok gry

Gra polega na odbijaniu krążka celem zdobycia bramki przeciwnika. Krążek porusza się w środowisku niewielkiego tarcia. Paletka gracza może poruszać się tylko w obszarze własnej połowy. Każda runda rozpoczyna się od umiejscowienia krążka w centrum areny. Gra kończy się po zdobyciu 10 punktów jednak z wymaganą przewagą 2 punktów.



Rysunek 8: Menu zakończonej gry

Po zakończonej grze możemy powrócić do menu lub zagrać ponownie.

Paint test

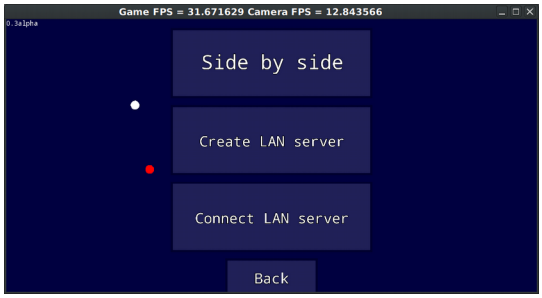
Paint test służy do testowania działania wskaźników. Zawiera przycisk czyszczenia płótna. Aby przejść do menu głównego należy wcisnąć przycisk Escape.

Rysunek 9: Widok paint testu

Menu gry wieloosobowej

Z menu gry wieloosobowej możemy przejść do:

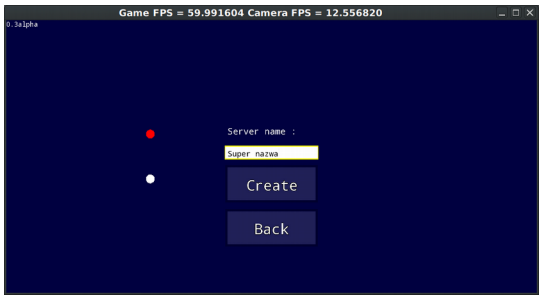
- Gry "side by side" - czyli granie przy jednym komputerze dla 2 osób
- Utworzenie serwera LAN
- Połączenie z serwerem LAN



Rysunek 10: Menu gry wieloosobowej

Gra przez LAN

W menu tworzenia serwera możemy ustawić nazwę serwera. Następnie możemy przejść do lobby lub powrócić do wcześniejszego menu.



Rysunek 11: Menu tworzenia serwera

Rysunek 12: Lobby po stronie serwera

Obrazek przedstawia widok lobby po stronie serwera. Przycisk play pojawia się wtedy gdy klient wybierze nasz serwer. Po kliknięciu play rozpoczyna się rozgrywka wieloosobowa.

Rysunek 13: Widok listy serwerów po stronie klienta

Ustawienia

Kalibracja kamerki i detektora

Na przedstawionym obrazku widać pierwszą stronę ustawień. Kamerka pracowała w słabym oświetleniu. Źródłem światła była jedynie lampka nocna lecz dzięki odpowiedniej konfiguracji detekcja punktów wskaźnika działa prawidłowo.

Rysunek 14: Pierwsza strona ustawień

Parametry do konfiguracji to:

- **Ekspozycja** - im wyższa tym dłużej naświetlana jest klatka. Powoduje zwiększenie jasności lecz przy tanich kamerkach także spadek FPS poniżej akceptowalnych wartości.
- **Gamma** - im wyższa tym ciemniejsze obszary są rozjaśniane, zbyt duża wartość powoduje utratę ogólnego kontrastu.
- **Balans bieli** - pozwala skompensować nadmierne przesunięcie barw w stronę czerwonego lub niebieskiego
- **Przesunięcie barwy** - pozwala dokonać dużego przesunięcia barw w przestrzeni HSV. Przydatne gdy wykrywany kolor jest w okolicach początku (czerwony) lub końca (fioletowy). Pozwoli to na lepsze dobranie składowej H niskiej oraz wysokiej
- **Składowa detektora H niska** - oznacza dolną granicę wykrywanej barwy w przestrzeni HSV
- **Składowa detektora H wysoka** - oznacza górną granicę wykrywanej barwy w przestrzeni HSV
- **Składowa detektora S niska** - oznacza dolną granicę wykrywanego nasycenia w przestrzeni HSV
- **Składowa detektora S wysoka** - oznacza górną granicę wykrywanego nasycenia w przestrzeni HSV
- **Składowa detektora V niska** - oznacza dolną granicę wykrywanej jasności w przestrzeni HSV
- **Składowa detektora V wysoka** - oznacza górną granicę wykrywanej jasności w przestrzeni HSV
- **Próg ruchu** - służy do konfiguracji maski ruchu. Im wyższa wartość tym różnica klatek aktualnej i poprzedniej musi być większa aby maska zadziałała w tych punktach.
- **Próg maski okręgu 1 oraz 2** - służy do konfiguracji maski detektora okręgów. Niska wartość powoduje wykrycie większej ilości okręgów. Może powodować spowolnienie procesu przetwarzania.
- **Minimalny rozmiar** - oznacza minimalny rozmiar wykrywanego punktu. Przydatne do odfiltrowania punktowych szumów mniejszych od właściwego znacznika a które nie mogą zostać wyeliminowane w inny sposób.
- **Limit punktów** - ogranicza nadmierną ilość wykrytych punktów. Do prawidłowego działania wystarczą 4.

Kalibracja polega na ręcznym ustawianiu parametrów metodą prób i błędów aż do osiągnięcia optymalnych wartości.

Druga strona ustawień zawiera następujące opcje:

Rysunek 15: Druga strona ustawień

- Przycisk przełączający wyświetlanie obrazu z kamery w tle
- Przycisk przełączający kolisty obszar kliknięcia wskaźnika
- Przycisk przełączający widok wskaźników w grze
- Przycisk do testowania algorytmu paletki

Plik konfiguracyjny

Przykładowa zawartość pliku konfiguracyjnego:

deviceId 0 #Id urządzenia przechwytującego, ręczna konfiguracja może być przydatna gdy jest podłączonych kilka urządzeń przechwytujących obraz.
MaxId 5 #Maksymalny id dla pętli auto-wykrywania. Zmiana może być przydatna przy nietypowo wysokim identyfikatorze urządzenia przechwytującego
exposure 299 #Ekspozycja
gamma 180 #Korekcja gamma
whiteBalance 5001 #Balans bieli
hueShift 90 #Przesunięcie barwy
hueLow 80 #Składowa H niska
hueHigh 110 #Składowa H wysoka
saturationLow 55 #Składowa S niska
saturationHigh 255 #Składowa S wysoka
valueLow 65 #Składowa V niska
valueHigh 255 #Składowa V wysoka
moveThreshold 15 #Próg ruchu
circleMaskThreshold1 177 #Próg maski okręgu 1
circleMaskThreshold2 88 #Próg maski okręgu 2
pointsLimit 4 #Limit punktów
minimumSize 18.548212 #Minimalny rozmiar
serverName Uw== #Nazwa serwera zakodowana w base64
winSizeX 1039 #Szerokość okna
winSizeY 644 #Wysokość okna
winPosX 0 #Pozycja X okna
winPosY 0 #Pozycja Y okna
leftNick QQ== #Lewy nick zakodowany w base64
rightNick Qg== #Prawy nick zakodowany w base64
pointerClickCircle 1 #Wyświetlanie promienia kliknięcia
pointerInGame 1 #Wyświetlanie wskaźników w grze
videoBackground 1 #Wyświetlanie obrazu z kamerki w grze

Dokumentacja techniczna

Wykorzystane narzędzia

Biblioteki:

- SFML
- OpenCV

Programowanie:

- Eclipse CDT
- Microsoft Visual Studio
- make
- git

Tworzenie dokumentacji:

- doxygen
- medit
- Libre Office Writer

Tworzenie prezentacji:

- Inkscape
- Microsoft PowerPoint

Założenia programu

Zrealizowane

- Działanie na platformie Windows oraz Linux dzięki SFML i OpenCV
- Implementacja algorytmu wykrywania wskaźników z wykorzystaniem biblioteki OpenCV
- Paint test
- Gra Air Hockey
  - Tryb gry jednoosobowej z komputerowym przeciwnikiem
  - Tryb gry wieloosobowej w trybie "side by side" (na jednym komputerze)
  - Podstawowy tryb gry w sieci lokalnej
  - Podstawowa fizyka odbicia krążka uwzględniająca kąt paletki i wartość bezwzględną prędkości liniowej
  - Automatycznie skalowana arena do rozmiaru okna

- Interfejs graficzny
  - Automatycznie dopasowujące i skalowane menu do okna aplikacji
  - Przycisk z możliwością przypisania akcji jako obiekt std::function<void(const MenuButton&)>
  - Menu zbudowane w oparciu o kontener przycisków
  - Jednoliniowe wejściowe pole tekstowe
  - Suwak do nastawy wartości liczbowych, z możliwością ustawienia wskaźnika na docelową zmienną
- Ustawienia
  - Zapamiętywanie konfiguracji w pliku tekstowym z łatwą możliwością rozbudowy o kolejne parametry
  - Możliwość prostego ustawienia parametrów kamerki
  - Możliwość prostego ustawienia parametrów detektora punktów

Nie zrealizowane

- Gra Air Hockey
  - Zaawansowana fizyka odbicia krążka uwzględniająca precyzyjnie wektor prędkości liniowej oraz prędkość kątową
  - Możliwość zmiany strony gry w trybie jednoosobowym oraz wieloosobowym sieciowym
  - Możliwość zmiany ilości potrzebnych punktów do zdobycia
  - Możliwość przełączania gry z przewagą 2 punktów
  - Zaawansowane lobby z funkcją czatu i ustawień rozgrywki
- Interfejs graficzny
  - Łatwiejszy mechanizm przełączania widoków menu
  - Menu oparte o kontener abstrakcyjnego typu kontrolki

Kompilacja

Wymagania

Aby skompilować projekt należy posiadać następujące biblioteki:

- SFML >= 2.3.3
- OpenCV >= 3.2
- Video4Linux (tylko dla dystrybucji Linux)

Opcjonalnie można zainstalować **doxygen** do generacji dokumentacji html.

Linux

Zalecana jest kompilacja z użyciem narzędzia **make**, które należy zainstalować jeżeli nie jest dostępne.

Dodatkowo wymagany jest kompilator **C++** obsługujący standard **C++11**. Zaleca się użycie **g++6**.

Instalacja pakietów dla dystrybucji wywodzących się z Debiana:

sudo apt-get install build-essential libsfml-dev libv4l-dev libopencv-dev

Jeżeli wersja biblioteki OpenCV w repozytorium jest nieodpowiednia należy przeprowadzić kompilacje biblioteki ze źródeł OpenCV GIT.

Po udanej kompilacji należy pamiętać o uruchomieniu komendy ldconfig jako root.

Następnie należy przejść do katalogu z kodem źródłowym:

cd NAZWA_KATALOGU_ŹRÓDŁOWEGO

make PH=off ARG=-O3

Użycie PH=off powoduje że nie zostaną utworzone prekompilowane nagłówki natomiast ARG=-O3 przekazuje dodatkową flagę optymalizacji 3 stopnia do kompilatora.

Po udanej kompilacji powinien zostać wygenerowany plik binarny **airhockey**

Można go uruchomić z terminala, znajdując się w tym samym katalogu:

./airhockey

Dodatkowe możliwości pliku makefile:

Usuwanie plików *.o

make clean

Usuwanie plików *.o oraz prekompilowane nagłówki *.gch i pomocnicze *.md5*

make cleanall

Import projektu do Eclipse CDT

Wybieramy:

File > Import... > C/C++ > Existing code as Makefile Project

Wybieramy lokacje kodu źródłowego oraz zaznaczamy język **C++** oraz Toolchain for Indexer Settings na **Linux GCC**

Po imporcie ustawiamy plik wykonywalny

Run > Run Configurations... > C/C++ Application > (PPM) New

Następnie wpisujemy nazwę pliku **airhockey.**

Możemy także usprawnić naszą kompilacje przez użycie wielu rdzeni:

Projekt > Properties > C/C++ Build

Odznaczamy Use default build command i dla 4 rdzeni wpisujemy w build commmand:

make -j4

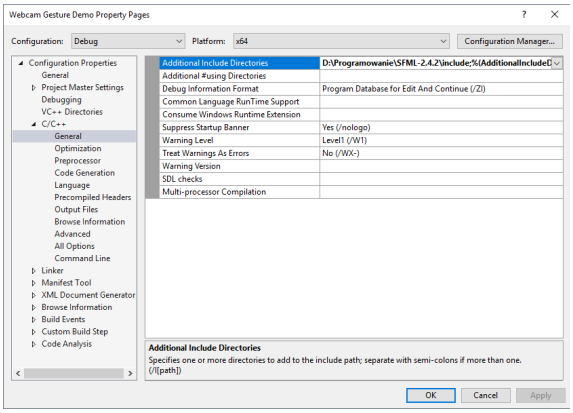
Windows

Kompilacja dla systemów Windows może być przeprowadzona w Visual Studio.

Na początku należy utworzyć nowy pusty projekt **C++** a następnie przejść do ustawień:

Project > NAZWA_PROJEKTU properties > C/C++ > General

W Additional Include Directories dodajemy katalog include z SFML.



Rysunek 16: C/C++ General

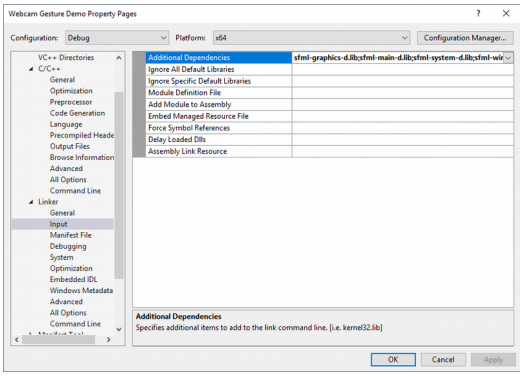
Następnie przechodzimy do ustawień:

Linker > Input

W Additional Dependencies należy ustawić następujące pliki:

sfml-graphics-d.lib
sfml-main-d.lib
sfml-system-d.lib
sfml-window-d.lib
sfml-network-d.lib

Dla kompilacji release należy wybrać analogiczne pliki bez końcówki -d

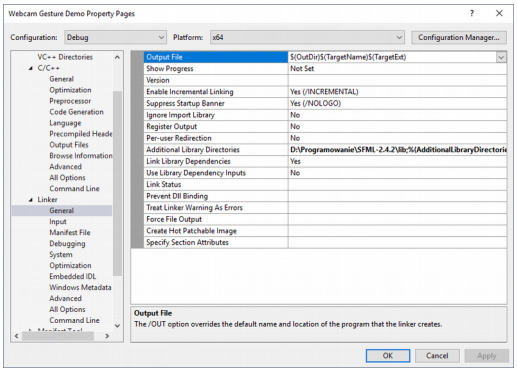


Rysunek 17: Linker input

Dalej przechodzimy do ustawień:

Linker > General

W Additional Library Directories dodajemy katalog lib z SFML.



Rysunek 18: Linker General

Bibliotekę OpenCV można zainstalować przez NuGet:

Tools > NuGet Package Manager > Package Manager Console

Wpisujemy komendę:

Install-Package OpenCV -Version 2.4.11

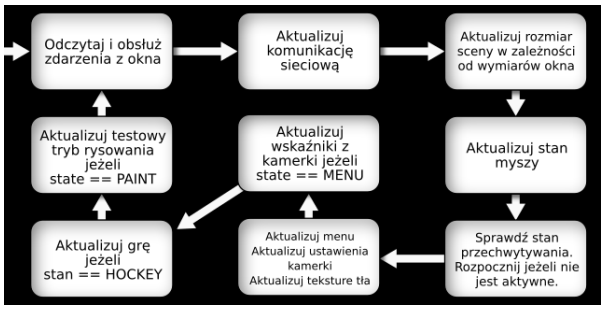
Działanie programu

Aplikacja działa w oparciu o pętlę programu wykonywaną w wątku głównym a także pętlę wykonywaną w wątku pomocniczym.

Decyzja o zastosowaniu dodatkowego wątku została podjęta z uwagi na fakt, że odczyt ramki z kamerki internetowej odbywa się w stosunkowo długim czasie co mogłoby spowodować utratę płynności działania aplikacji.

Uproszczony diagram głównej pętli

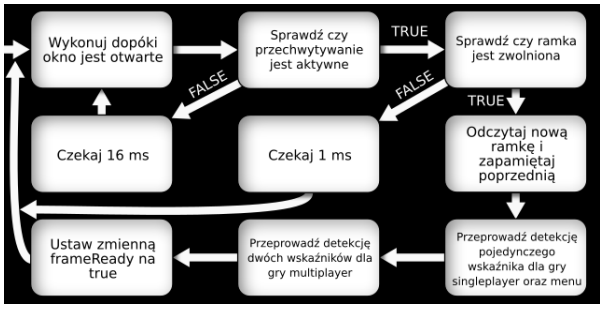
Główny wątek odpowiada za odczytywanie zdarzeń z okna i reakcją na nie. Dokonywane jest nawiązanie połączenia z kamerką internetową i nastawa jej parametrów. Dodatkowo w nim są aktualizowane wszystkie obiekty potrzebne w aktualnym widoku. Jest tam między innymi obsługa elementów interfejsu graficznego jak i obsługa komunikacji sieciowej. Ostatnią czynnością jest wyrysowanie wszystkich obiektów na scenie i wyświetlenie wygenerowanej klatki.



Rysunek 19: Diagram pętli głównej

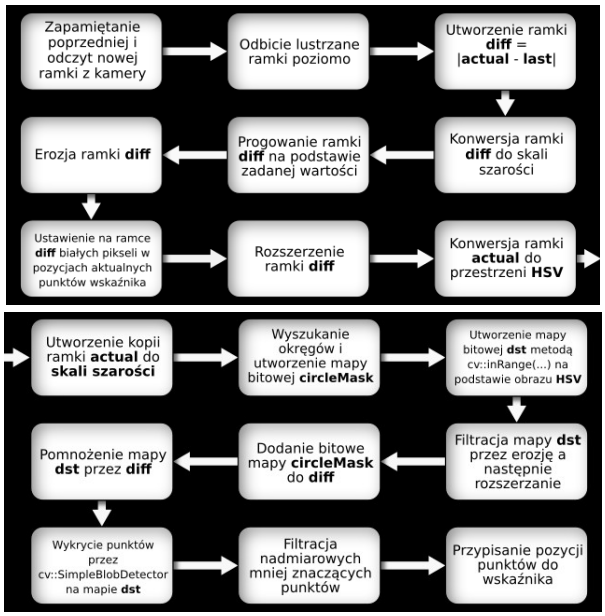
Uproszczony diagram pętli wątku pomocniczego

Pomocniczy wątek odpowiada za odczytanie klatki z urządzenia przechwytującego a następnie przeprowadzenie detekcji punktów wskaźnika lub dwóch wskaźników w zależności od stanu aplikacji.



Rysunek 20: Diagram pętli pomocniczej

Diagram algorytmu detekcji wskaźników



Rysunek 21: Diagram detekcji wskaźników

Na powyższych rysunkach przedstawiono w uproszczony sposób algorytm detekcji wskaźników. Metody OpenCV warte uwagi:

- cv::absdiff - do wyznaczania absolutnej różnicy pomiędzy dwoma ramkami
- cv::cvtColor - do konwersji przestrzeni barw
- cv::threshold - do progowania ramki
- cv::erode - do operacji erozji
- cv::dilate - do operacji rozszerzania
- cv::bitwise_or - do bitowej sumy dwóch ramek
- cv::bitwise_and - do bitowego iloczynu dwóch ramek
- cv::inRange - do selekcji kolorów
- cv::HoughCircles - do wykrywania okręgów
- cv::SimpleBlobDetector - do do wykrywania jednobarwnych plam

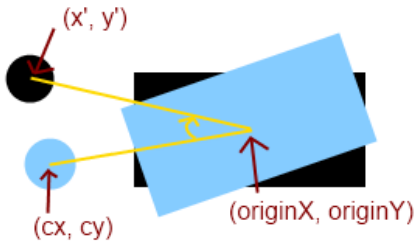
Detekcja kolizji krążka z paletką

Detekcja kolizji krążka z obróconą paletką może być nieco problematyczna. Istnieje natomiast prosty sposób jak to osiągnąć. Zamiast obracać prostokąt (paletkę) należy obrócić krążek pod tym samym kątem pod którym obrócilibyśmy paletkę. Możemy to osiągnąć za pomocą następujących wzorów:

$$x' = \mathbf{cos}(theta) * (cx - originX) - \mathbf{sin}(theta) * (cy - originY) + originX$$

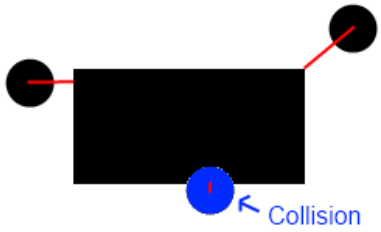
$$y' = \mathbf{sin}(theta) * (cx - originX) + \mathbf{cos}(theta) * (cy - originY) + originY$$

Na poniższym rysunku został przedstawiony ten proces. Niebieskie elementy to te które gracz widzi na ekranie, natomiast obliczenia są przeprowadzane dla elementów o kolorze czarnym.



Rysunek 22: Obracanie krążka

Następnie za pomocą prostych instrukcji warunkowych należy znaleźć punkt na paletce, który jest najbliżej środka krążka. Ostatnim krokiem jest sprawdzenie odległości między tym punktem, a środkiem krążka za pomocą twierdzenia Pitagorasa. Jeżeli odległość ta jest mniejsza od promienia krążka, to oznacza, że nastąpiła kolizja.



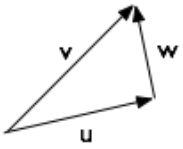
Rysunek 23: Kolizja

Obliczanie kąta odbicia

Do obliczenia nowego kąta krążka po odbiciu zastosowaliśmy obliczenia wektorowe. Kąt pod którym leci krążek rozbijamy na postać wektorową, na składową x oraz y.

x = **cos**(angle);

y = **cos**(angle);



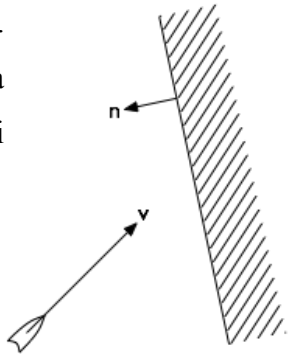
Rysunek 24

Następnie tworzymy 2 nowe wektory. Wektor u, który jest prostopadły do obiektu, z którym nastąpiła kolizja oraz wektor w równoległy do tej ściany. Te wektory uzyskujemy za pomocą wzorów:

$$u = (v \cdot n / n \cdot n) * n$$

$$w = v - u$$

Gdzie n oznacza wektor siły prostopadłej do ściany. Posiadając już te 2 wektory możemy uzyskać wektor krążka po odbiciu (v' = w - u) oraz przekształcić go do postaci kątowej.

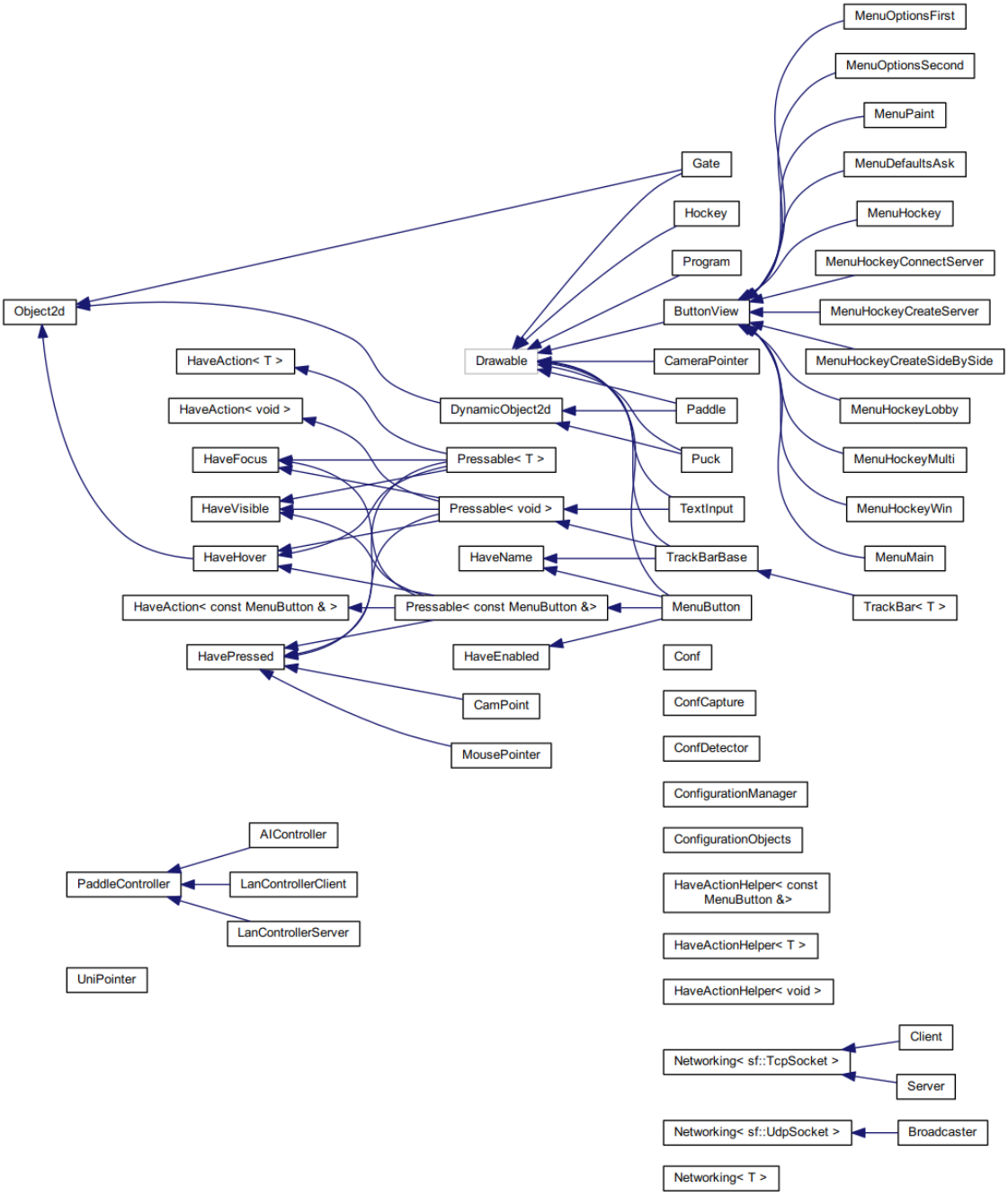


Rysunek 25

Lista nagłówków *.hpp

AIController.hpp	Definicja klasy **AIController** dziedziczącej po **PaddleController**
Broadcaster.hpp	Definicja klasy **Broadcaster** dziedziczącej po **Networking**
ButtonView.hpp	Definicja klasy **ButtonView** dziedziczącej po sf::Drawable
CameraPointer.hpp	Definicja klasy **CameraPointer** dziedziczącej po sf::Drawable oraz klasy **CamPoint** dziedziczącej po **HavePressed**
Client.hpp	Definicja klasy **Client** dziedziczącej po **Networking**
Conf.hpp	Definicja statycznej klasy **Conf**
ConfCapture.hpp	Definicja statycznej klasy **ConfCapture**
ConfDetector.hpp	Definicja statycznej klasy **ConfDetector**
ConfigurationManager.hpp	Definicja klasy **ConfigurationManager** oraz struktury **ConfigurationObjects**
DynamicObject2d.hpp	Definicja bazowej klasy **DynamicObject2d** dziedziczącej po **Object2d**
Gate.hpp	Definicja klasy **Gate** dziedziczącej po public sf::Drawable i **Object2d**
HaveAction.hpp	Definicja bazowej generycznej klasy **HaveAction**
HaveEnabled.hpp	Definicja bazowej klasy **HaveEnabled**
HaveFocus.hpp	Definicja bazowej klasy **HaveFocus**
HaveHover.hpp	Definicja bazowej klasy **HaveHover** dziedziczącej po **Object2d**
HaveName.hpp	Definicja bazowej klasy **HaveName**
HavePressed.hpp	Definicja bazowej klasy **HavePressed**
HaveVisible.hpp	Definicja bazowej klasy **HaveVisible**
Hockey.hpp	Definicja klasy **Hockey** dziedzicząca po sf::Drawable
LanControllerClient.hpp	Definicja klasy **LanControllerClient** dziedzicząca po **PaddleController**
LanControllerServer.hpp	Definicja klasy **LanControllerServer** dziedzicząca po **PaddleController**
MenuButton.hpp	Definicja klasy **MenuButton** dziedziczącej po sf::Drawable, **Pressable**, **HaveName** i **HaveEnabled**
MenuDefaultsAsk.hpp	Definicja klasy **MenuDefaultsAsk** dziedziczącej po **ButtonView**
MenuHockey.hpp	Definicja klasy **MenuHockey** dziedziczącej po **ButtonView**
MenuHockeyConnectServer.hpp	Definicja klasy **MenuHockeyConnectServer** dziedziczącej po **ButtonView**
MenuHockeyCreateServer.hpp	Definicja klasy **MenuHockeyCreateServer** dziedziczącej po **ButtonView**
MenuHockeyCreateSideBySide.hpp	Definicja klasy **MenuHockeyCreateSideBySide** dziedziczącej po **ButtonView**
MenuHockeyLobby.hpp	Definicja klasy **MenuHockeyLobby** dziedziczącej po **ButtonView**
MenuHockeyMulti.hpp	Definicja klasy **MenuHockeyMulti** dziedziczącej po **ButtonView**
MenuHockeyWin.hpp	Definicja klasy **MenuHockeyWin** dziedziczącej po **ButtonView**
MenuMain.hpp	Definicja klasy **MenuMain** dziedziczącej po **ButtonView**
MenuOptionsFirst.hpp	Definicja klasy **MenuOptionsFirst** dziedziczącej po **ButtonView**
MenuOptionsSecond.hpp	Definicja klasy **MenuOptionsSecond** dziedziczącej po **ButtonView**
MenuPaint.hpp	Definicja klasy **MenuPaint** dziedziczącej po **ButtonView**
MousePointer.hpp	Definicja klasy **MousePointer** dziedziczącej po **HavePressed**
Networking.hpp	Definicja generycznej klasy **Networking**
Object2d.hpp	Definicja klasy **Object2d**
Paddle.hpp	Definicja klasy **Paddle** dziedziczącej po sf::Drawable i **DynamicObject2d**
PaddleController.hpp	Definicja klasy **PaddleController**
Pressable.hpp	Definicja bazowej klasy **Pressable** dziedziczącej po **HaveAction**, **HaveHover**, **HavePressed**, **HaveVisible** i **HaveFocus**
Program.hpp	Definicja klasy **Program** dziedziczącej po sf::Drawable
Puck.hpp	Definicja klasy **Puck** dziedziczącej po sf::Drawable i **DynamicObject2d**
S.hpp	Definicja przestrzeni nazw S która zawiera stałe liteały tekstowe
Server.hpp	Definicja klasy **Server** dziedziczącej po **Networking**
TextInput.hpp	Definicja klasy **TextInput** dziedziczącej po sf::Drawable i **Pressable**
TrackBar.hpp	Definicja klasy **TrackBarBase** dziedziczącej po sf::Drawable, **HaveFocus** i **Pressable** oraz definicja klasy **TrackBar** dziedziczącej po **TrackBarBase**
UniCameraPointer.hpp	Definicja klasy **UniPointer**

Diagram klas



Lista klas

AIController	Klasa kontrolera paletki. Służy do sterowania paletką przez prosty autorski algorytm.
Broadcaster	Klasa broadcastera UDP. Służy do rozgłaszania pakietów UDP w sieci lokalnej. Rozgłaszanie jest aktywowane gdy stworzymy nowy serwer i dezaktywuje się zaraz po połączeniu klienta.
ButtonView	Bazowa klasa widoku z przyciskami. Można dodawać do niego przyciski klasy MenuButton. Należy implementować metodę update() która odpowiada za pozycjonowanie zawartych przycisków.
CameraPointer	Klasa wskaźnika sterowanego przez kamerkę. Reprezentuje wskaźnik sterowany przez kamerkę, który składa się z dwóch punktów klasy CamPoint. Wskaźnik ten zawiera w sobie instancję cv::SimpleBlobDetector która pozwala wykrywać punkty z ramki cv::Mat. Wszelkie parametry służące do konfiguracji procesu wykrywania punktów są odczytywane ze statycznej klasy Conf.
CamPoint	Obiekty tej klasy są zawarte w klasie **CameraPointer.** Reprezentuje pojedynczy punkt. Zawiera odpowiednie pola aktualnego stanu oraz poprzedniego.
Client	Klasa klienta gry. Służy do nawiązania połączenia z serwerem gry Air Hockey.
Conf	Statyczna klasa konfiguracji. Służy do globalnego dostępu dla wszystkich innych klas. Publicznie dostępne pola stanowią głównie stałe oraz stałe referencje do pól prywatnych. Klasa ta jest zaprzyjaźniona z klasą Program oraz ConfigurationManager, przez co mogą zmieniać jej stan.
ConfCapture	Statyczna klasa konfiguracji kamerki. Publicznie dostępne są stałe referencje do czasu odczytania ramki oraz przesunięcia barwy. Klasa ta jest zaprzyjaźniona z klasą Program oraz ConfigurationManager, przez co mogą zmieniać jej stan.
ConfDetector	Statyczna klasa konfiguracji detektora. Zawiera konfigurację używaną w procesie wykrywania punktów. Klasa ta jest zaprzyjaźniona z klasą Program oraz ConfigurationManager, przez co mogą zmieniać jej stan.
ConfigurationManager	Klasa menedżera konfiguracji służy do zapisywania i odczytywania istotnych parametrów do pliku tekstowego program.conf.
ConfigurationObjects	Struktura zawiera referencje na dodatkowe konfigurowane obiekty.
DynamicObject2d	Bazowa klasa dynamicznego obiektu. Posiada pola dotyczące prędkości liniowej i obrotowej.
Gate	Klasa bramki w grze
HaveAction	Bazowa klasa obiektu z ustawialną akcją typu std::function<void(T)>
HaveActionHelper	Struktura pomocnicza dla argumentów typów != void
HaveActionHelper< void >	Struktura pomocnicza dla argumentu typu == void
HaveEnabled	Bazowa klasa obiektu włączalnego
HaveFocus	Bazowa klasa obiektu "focusowalnego". Reprezentuje obiekt posiadający pole focus. Poprzednio skupiony obiekt jest zapamiętany w statycznym polu, dzięki temu po wywołaniu metody setFocus(true) stan focus poprzedniego obiektu jest ustawiany na false.
HaveHover	Bazowa klasa obiektu posiadającego stan wskazywania przez kursor
HaveName	Bazowa klasa obiektu nazywalnego
HavePressed	Bazowa klasa obiektu posiadający stan wciśnięcia
HaveVisible	Bazowa klasa obiektu posiadający stan widoczności
Hockey	Klasa gry Air Hockey. Zawiera między innymi obiekty klas Paddle, Puck, Gate, PaddleController.
LanControllerClient	Klasa sieciowego kontrolera paletki przeciwnika po stronie klienta. Odbiera pozycje wskaźnika z serwera oraz pozycje krążka. Wysyła dane paletki klienta.
LanControllerServer	Klasa sieciowego kontrolera paletki przeciwnika po stronie serwera. Odbiera pozycje wskaźnika od klienta. Wysyła dane paletki serwera oraz pozycje krążka.
MenuButton	Klasa przycisku menu
MenuDefaultsAsk	Klasa reprezentująca widok menu z pytaniem o przywrócenie domyślnych ustawień
MenuHockey	Klasa reprezentująca widok menu wyboru trybu singleplayer i multiplayer

MenuHockeyConnectServer	Klasa reprezentująca widok menu wyboru serwera
MenuHockeyCreateServer	Klasa reprezentująca widok menu tworzenia serwera
MenuHockeyCreateSideBySide	Klasa reprezentująca widok menu tworzenia gry side by side
MenuHockeyLobby	Klasa reprezentująca widok lobby serwera oczekującego na gracza
MenuHockeyMulti	Klasa reprezentująca widok menu wyboru trybu side by side, tworzenia serwera i łączenia z serwerem
MenuHockeyWin	Klasa reprezentująca widok menu wygranej gry z możliwością ponownej rozgrywki
MenuMain	Klasa reprezentująca widok menu głównego widocznego po uruchomieniu programu
MenuOptionsFirst	Klasa reprezentująca menu pierwszej strony opcji
MenuOptionsSecond	Klasa reprezentująca menu drugiej strony opcji
MenuPaint	Klasa reprezentująca menu paint testu z przyciskiem do czyszczenia płótna
MousePointer	Klasa reprezentująca mysz. Zwiększa funkcjonalność standardowej klasy sf::Mouse
Networking	Abstrakcyjna klasa bazowa dla **Client** i **Server**
Object2d	Bazowa klasa obiektów posiadających pozycję, rozmiar i kąt obrotu
Paddle	Klasa reprezentująca paletkę w grze Air **Hockey**
PaddleController	Abstrakcyjna bazowa klasa kontrolera paletki. Jest używana w klasie **Hockey**
Pressable	Bazowa klasa obiektu wciskalnego
Program	Główna klasa aplikacji. Steruje przepływem informacji. Zawiera wszystkie widoki menu, inne elementy interfejsu oraz klasy odpowiedzialne za grę oraz komunikacje sieciową. Aktualizuje swój stan oraz innych obiektów w reakcji na określone zdarzenia.
Puck	Klasa reprezentuje krążek hokejowy
Server	Klasa służąca do nawiązania połączenia oraz komunikacji z klientem
TextInput	Klasa reprezentująca jednoliniowe edytowalne pole tekstowe
TrackBar	Generyczna klasa **TrackBar**'a. Umożliwia tworzenie track bara dla różnych typów numerycznych. Po ustawieniu wskaźnika na modyfikowaną zmienną, zmiana pozycji track bara automatycznie przepisze do niej swoją wartość.
TrackBarBase	Bazowa klasa **TrackBar**'a. Zapewnia reprezentację i funkcjonalność track bara do ustawiania wartości pomiędzy wartością minimalną a maksymalną.
UniPointer	Klasa uniwersalnego wskaźnika. Zwraca stan **MousePointer** lub **CameraPointer.** Zawiera referencje na MousePointer i CameraPointer. Kiedy aktywna jest mysz, metody getPos() oraz isPressed() zwracają stan myszki. W przeciwnym wypadku zwracany jest stan wskaźnika obsługiwanego przez kamerkę.

Statystyka kodu źródłowego

Podstawowa statystyka została wyliczona przy użyciu następującego ciągu komend:

cat *pp | **uniq** | **wc**

- **6875** linii
- **15520** ciągów
- **162544** znaków

Harmonogram

- 06.03 – 19.03 Zapoznanie się z tematem
- 20.03 – 31.03 Prototyp wykrywania punktów
- 01.04 – 07.04 Prototyp Air Hockeya
- 01.04 – 14.04 Tworzenie projektu z komunikacją sieciową
- 07.04 – 14.04 Rozbudowa metody wykrywania punków, GUI
- 14.04 – 21.04 Rozbudowa gry, implementacja painta
- 21.04 – 01.05 Dostosowanie komunikacji sieciowej do rozbudowanej wersji gry
- 01.05 – 14.05 Scalenie projektów komunikacji z projektem Air Hockeya
- 14.05 – 10.06 Refaktoryzacja kodu, ostatnie poprawki

Podział prac

- **Jakub Mendel** – szkielet aplikacji, przetwarzanie obrazu
- **Piotr Okoń –** logika i wygląd gry
- **Grzegorz Kokoszka –** komunikacja sieciowa
- **Kamil Wojciechowski –** komunikacja sieciowa

Wnioski

Główny cel czyli zastosowanie kamerki internetowej oraz jednokolorowych wskaźników jako kontroler do gry został osiągnięty. Niestety nie udało się zrealizować wszystkich założeń z uwagi na nieprecyzyjne oszacowanie czasu. Wynikało ono głównie z potrzeby pogłębienia wiedzy w poruszanych zagadnieniach związanych z przetwarzaniem obrazu oraz komunikacji sieciowej w czasie rzeczywistym. Ważnym wnioskiem jest fakt, że dzięki zastosowaniu biblioteki OpenCV zwyczajny komputer, wyposażony w kamerkę internetową, jest w stanie przetwarzać obraz w czasie rzeczywistym przy częstotliwości 30 Hz. Mnogość poradników i kursów w sieci tej biblioteki pozwalają każdemu adeptowi programowania z niej skorzystać, natomiast jej ogromne możliwości dają duży potencjał dla nowatorskich rozwiązań. Doświadczenie zdobyte w tym projekcie z pewnością zaprocentuje w przyszłości.