Dokumentacja projektu

Podstawy teleinformatyki

Prowadzący: mgr inż. Przemysław Walkowiak

Temat projektu:

Rozpoznawanie obrazu z gry Warcaby oraz wizualizacja stanu gry na komputerze

Rafał Adamski Marcin Sznura Tomasz Weiss Michał Wypchło

Informatyka, Semestr VII

Studia niestacjonarne

18 stycznia 2020

Spis treści

1.Charakterystyka projektu

1.1 Opis projektu

Przedmiotem niniejszego projektu jest zaprojektowanie oraz implementacja systemu, którego zadaniem jest odczyt oraz przetwarzanie informacji o przebiegu rozgrywki z gry planszowej Warcaby na podstawie danych dostarczanych przez obraz wideo z kamery internetowej, która powinna być umieszczona nad planszą. Program ma za zadanie rozpoznać poszczególne figury - ich kolor, a także rodzaj(pionek czy damka) - na planszy wraz z ich wykonywanymi akcjami, a następnie za pomocą pozostałych komponentów w postaci cyfrowej zwizualizować stan gry na komputerze, jednocześnie w trakcie trwania rozgrywki weryfikując poprawność wykonywanych na planszy ruchów, czy też bić i informować użytkownika w przypadku błędnych, czy też nielegalnych operacji.

System może być idealnym narzędziem do nauki gry w Warcaby, pozwoli on na przyswojenie zasad rozgrywki, mógłby także być ciekawym rozwiązaniem do transmisji rozgrywek na żywo.

1.2 Uzasadnienie wyboru tematu

Zgodnie z całym zespołem wybraliśmy ten temat projektu, ponieważ w dzisiejszych czasach rejestracja obrazu, jego przetwarzanie i analiza stają się wszechobecne, tego typu operacje stosowane są w wielu dziedzinach, takich jak medycyna, kryminologia, ale także rozrywce dostępnej dla każdego - co za tym idzie ma realny wpływ na nasze życie, zdrowie, komfort czy też bezpieczeństwo. Oprócz samego przetwarzania obrazu projekt łączy w sobie również tematykę gier oraz implementację zasad i reguł gry, co wzbudziło nasze zainteresowanie. Uznaliśmy to za idealną okazję do zapoznania się z nowymi mechanizmami, z którymi wcześniej nie mieliśmy do czynienia - żaden inny przedmiot nie dawał nam takich możliwości.

1.3 Podział prac i systemu na komponenty

Z uwagi na złożoność projektu, konieczne jest wyodrębnienie na etapie projektowania modułów, z których powinna składać się końcowa aplikacja. Projekt będzie składać się z następujących modułów:

- Moduł odpowiedzialny za przechwytywanie obrazu klatka po klatce z kamery internetowej,
- Moduł odpowiedzialny za wizualizację przechwytywanych informacji na temat planszy i pionków,
- Moduł odpowiedzialny za weryfikację poprawności wykonywanych akcji, zasad gry.

Projekt zespołowy niezależnie od stopnia trudności, wymaga podzielenia obowiązków między członków zespołu. Aby praca była efektywna, warto dostosować poziom zadań do umiejętności każdej z osób, preferencji, czy też predyspozycji.
Ze względu na powyższe czynniki rozdzieliliśmy zadania w następujący sposób:

Rafał Adamski	- Opracowanie zasad legalności ruchów dla poruszania się pionków (przy projekcie z C++, który w późniejszej fazie został porzucony), - Pomoc przy algorytmach sprawdzających legalność ruchów umieszczonych w skryptach Unity, - Prace nad dokumentacją projektową.
Marcin Sznura	- Zaprojektowanie oraz implementacja algorytmu przechwytywania oraz interpretacji obrazu z kamery internetowej w środowisku Unity, - Implementacja zasad gry z uwzględnieniem błędów wiążących się z przetwarzaniem obrazu - Implementacja wykrywania położenia planszy

Tomasz Weiss	- Przygotowanie planszy, - Testy działania logiki aplikacji, - Wyszukiwanie błędów.
Michał Wypchło	- Zaprojektowanie modeli obiektów 3D w aplikacji Blender, - Prace nad układem sceny w Unity, - Prace związane z interfejsem użytkownika, - Utworzenie skryptów z dodatkowymi funkcjonalnościami, - Modyfikacje tekstur obiektów, - Prace nad dokumentacją projektową.

2. Wymagania projektu

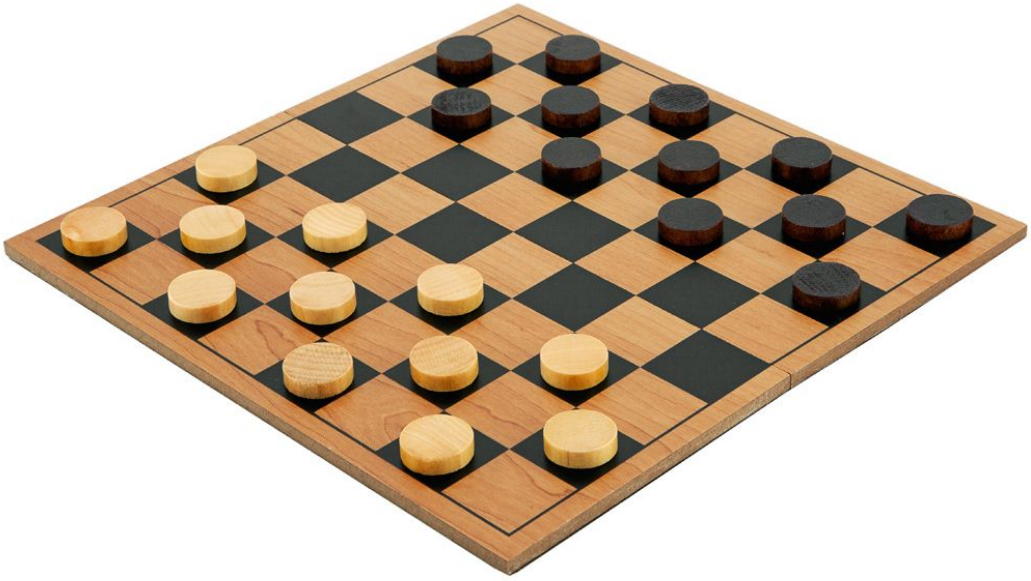
Określenie wymagań oprogramowania jest tym etapem projektu informatycznego, który jest konieczny w celu określenia jak powinien zachowywać się system w finalnej wersji.

2.1 Wymagania funkcjonalne

Aplikacja utworzona w ramach projektu z założenia powinna oferować następujące funkcjonalności:

1. Odczyt oraz przetworzenie obrazu z wybranej kamery
   1.1 Wybór kamery z wszystkich dostępnych w systemie
   1.2 Odczytanie pozycji pól planszy
   1.3 Odczytanie pozycji pionków białych oraz czarnych
   1.4 Odczytanie które spośród wykrytych pionków są damkami, a które zwykłymi pionkami.
2. Wizualizacja stanu gry na komputerze
   2.1 Sprawdzenie czy pionek znajduje się w maksymalnej możliwej odległości od pola (pionki poza tym zakresem traktowane są jako tło np. cienie na stole)
   2.2 Sprawdzenie do położenia którego pola, najbliżej jest położeniu pionka
   2.3 Wyświetlenie pionków na danych polach wizualizacji jeśli dla pól tych zostały spełnione warunki 2.1 oraz 2.2
3. Po wykonaniu ruchu pionka, program ma sprawdzać czy ruch został wykonany poprawnie.
   3.1 Sprawdzanie czy ilość pionków uległa poprawnej zmianie
      - dla ruchu bez bicia suma pionków obu graczy musi pozostać niezmieniona
      - dla ruchu z biciem: suma pionków gracza bijącego nie zmiania się, podczas gdy przeciwnik traci liczbę pionków równą ilości bić
   3.2 Sprawdzenie czy zostało wykonanie bicie jeśli możliwe
   3.3 Sprawdzenie czy możliwe bicie zostało wykonane poprawnie
   3.4 Sprawdzenie czy ruch pionka został wykonany poprawnie, jeśli nie było możliwego bicia

2.2 Zasady gry Warcaby



Istnieje kilka różnych odmian gry warcaby. Chociaż zdarzają się też różnice w ustawieniu pionków i zajmowanych polach(np. warcaby tureckie) to kluczowe są postanowienia dotyczące bicia pionków oraz ruchów damki. W naszym projekcie przyjęliśmy następujące zasady:

- Plansza 64 pola: 8x8
- 12 pionków białych i 12 czarnych
- Pionki ustawione na polach ciemnych
- Grę rozpoczyna domyślnie gracz czarny, lecz ustawienie to można zmienić
- Celem gry jest zbicie wszystkich pionków przeciwnika.
- Piony mogą poruszać się tylko do przodu (nie dotyczy bicia)

- Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne).
- Zbite piony są usuwane z planszy po zakończeniu ruchu. Jeśli istnieje możliwość wykonania bicia, to jest ono obowiązkowe. Jeśli po zbiciu pionka możemy bić dalej to dalsze bicie także jest obowiązkowe.
- Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, przy czym jeśli w jednym ruchu w wyniku wielokrotnego bicia przejdzie przez ostatni rząd, ale nie zakończy na nim ruchu, to nie staje się damką i kończy ruch jako pionek.
- Kiedy pion staje się damką, kolejny ruch przypada dla przeciwnika.
- Damki mogą poruszać się tak samo jak pionki z tą różnicą, że podczas normalnego ruchu bez bicia, mogą wykonać ruch do tyłu.

3. Wybrane technologie

Tworzenie aplikacji wymaga zgrania zespołu, ale także doboru odpowiednich narzędzi, które zostaną wykorzystane w procesie konstruowania systemu przez programistów oraz osoby zaangażowane. Odpowiednio dobrane technologie mogą znacząco wpłynąć na czas realizacji procesów, ale przede wszystkim wpływają na końcową jakość oprogramowania. W związku z preferencjami oraz naszym aktualnym doświadczeniem dobraliśmy następujące technologie:

- **Silnik Unity 3D w wersji 2018.3.7f1** - zintegrowane środowisko do tworzenia gier komputerowych oraz innych materiałów interaktywnych w trójwymiarze i dwuwymiarze. Unity to środowisko bardzo elastyczne i kompletne, co oznacza, że można w nim przejść przez wszystkie etapy budowania gry. Program dostarcza narzędzia o szerokich możliwościach, które można dostosować do własnych potrzeb i rozbudowywać własnymi wtyczkami. W kreatorze znajdziemy intuicyjne narzędzia do projektowania interfejsu gry, zaawansowane mechanizmy animacji i cieniowania modeli w oparciu o symulacje fizyczne, możliwość pisania skryptów w JavaScripcie i C#, a także rozbudowane profilowanie zużycia pamięci. W działaniu przypomina “kreator” gier na zasadzie Scratcha, jednak ma do zaoferowania o wiele więcej umożliwiając pisanie użytkownikowi skomplikowanych skryptów. Skrypty można pisać w 3 językach: C#, Javascript i Boo.
  By oszczędzić czas twórcom gier, Unity pozwala korzystać ze sklepu z zasobami bezpośrednio z programu. Znajdziemy tam różnego typu modele

dla gier 2D i 3D, a także rozszerzenia edytora, wtyczki, gotowe otoczenia, poziomy, plansze i wiele innych. Platforma Unity, na której należy założyć konto, by korzystać z programu, pozwala także przygotować gry do wyświetlania reklam i daje studiom dostęp do narzędzi analitycznych. Narzędzia Unity zostały zaprojektowane tak, by mogły z niego korzystać duże studia, niezależni deweloperzy i entuzjaści. Unity pozwala także przygotować grę do dystrybucji w wielu sklepach, w tym Microsoft Store, PSN czy Steam. Co ważne, pracując na Windowsie można stworzyć grę dla Linuksa, konsoli czy Androida.

- **Blender w wersji 2.81.1** - wolne i otwarte oprogramowanie do modelowania i renderowania obrazów oraz animacji trójwymiarowych. Jest narzędziem niezwykłym, z jednej strony minimalne rozmiary i dostępność za darmo, a z drugiej autentyczna konkurencyjność do takich pakietów jak 3DS MAX. Dzięki tej aplikacji, zabawa w 3D dostępna jest również dla osób dysponujących słabszym sprzętem. Blender oferuje naprawdę potężne możliwości, które nie sposób tutaj opisać (na łamach portalu znajduje się artykuł poświęcony tej aplikacji). Pierwszy kontakt z programem może wywołać jednak lekki szok gdyż interfejs użytkownika jest wysoce niestandardowy. Warto jednak nie załamywać rąk, praca równolegle z myszą i klawiaturą (nie jest to konieczne) jest naprawdę efektywna po osiągnięciu wprawy. Zaletą pakietu jest jego uniwersalność. Blender wyposażony został w narzędzia do modelowania 3d, renderingu oraz animacji. Program umożliwia tworzenie gier oraz aplikacji 3D. Dzięki dużej popularności w naszym kraju, można znaleźć sporo materiałów na jego temat dostępnych w języku polskim co nie jest również bez znaczenia. To rozwiązanie pozwoliło nam na zaprojektowanie modeli obiektów 3D, które następnie zostały wykorzystane w

aplikacji jako ‘Assety’, m.in. pionki(osobne dla zwykłego pionka oraz dla damki), czy też elementy planszy.

- **Język programowania C#** - spośród trzech języków, które oferuje Unity wybraliśmy język programowania C#. Jest on jednym z najbardziej powszechnych języków, mieliśmy z nim także najwięcej do czynienia zaraz po C++.
  Według wielu osób mających do czynienia z Unity, jest on najlepszym i jedynym potrzebnym językiem do pisania skryptów. Wynika to z faktu, że Unity używa Mono, która jest multiplatformową implementacją frameworku Microsoftu .NET i z tego względu wszystkie jego biblioteki są zbudowane na tym właśnie języku.

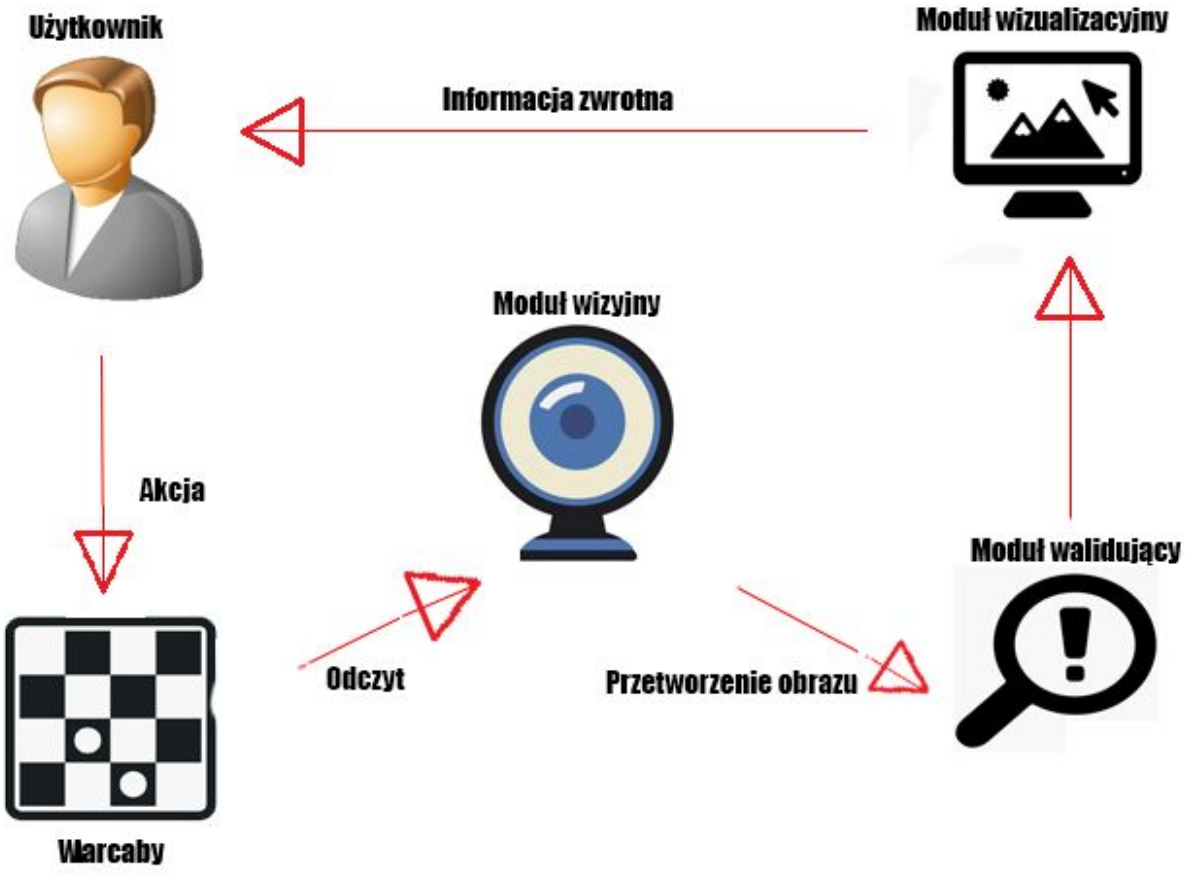
- **Środowisko programowe Visual Studio 2019 -** zintegrowane środowisko programistyczne firmy Microsoft. Jest używane do tworzenia oprogramowania konsolowego oraz z graficznym interfejsem użytkownika, w tym aplikacji

Windows Forms, WPF, Web Sites, Web Applications i inne.

- **GIT -** rozproszony system kontroli wersji, który został wybrany ze względu na powszechność w użyciu i prostotę w obsłudze. Projekt przechowywany jest na serwisie hostingowym **GitHub**, który wykorzystuje wspomniany system kontroli wersji.

4. Architektura rozwiązania

Budowę systemu można podzielić na trzy podstawowe komponenty, z których każdy odpowiedzialny jest za inną funkcjonalność:



Prosty schemat obrazujący działanie komponentów systemu

- Moduł wizyjny, który odczytuje stan planszy oraz pozycje pionków - jego rolę pełni obraz z kamerki internetowej, poddawany odpowiednim przekształceniom
- Moduł walidacyjny odpowiedzialny za weryfikację poprawności wykonanych ruchów oraz bić - tutaj główną rolę odgrywa skrypty “Game Master” w języku C# zintegrowane z naszym silnikiem aplikacji, który pilnuje przestrzegania zasad gry
- Moduł wizualizacyjny, który na podstawie informacji od modułu wizyjnego

obrazuje je na ekranie komputera - wykorzystujemy w tym celu stworzone przez nas obiekty 3D w środowisku Unity.

4.1 Architektura modułu wizyjnego

Jako że Unity nie posiada żadnych wbudowanych bibliotek czy dodatków do przetwarzania obrazu sami musieliśmy znaleźć sposób na stworzenie modułu wizyjnego. Podjeliśmy decyzję że nie skorzystamy z gotowych rozwiązań tj. np. biblioteka openCV, lecz sami zaimplementujemy skrypt napisany w c#, przetwarzający obraz na podstawie kolorów. Pomimo że takie rozwiązanie okazało się bardzo czasochłonne i nie pozbawione wad, udało nam się stworzyć zaprojektowany przez nas algorytm przetwarzania obrazu:

1. Zapisz obraz z kamery jako zdjęcie w formacie png.
2. Stwórz mapę zero-jedynkową z pixeli zdjęcia mieszczących się w zadanych granicach zakresów H, S i V.
3. Sąsiednie jedynki mapy połącz w grupy.
4. Grupy połącz ze sobą w obiekty jeśli znajdują się one bliżej siebie niż zadana wartość x.
5. Wyznacz skrajne wartości góra, dół, prawo, lewo uzyskanych obiektów.
6. Wyznacz środek z wartości góra, dół, prawo, lewo.

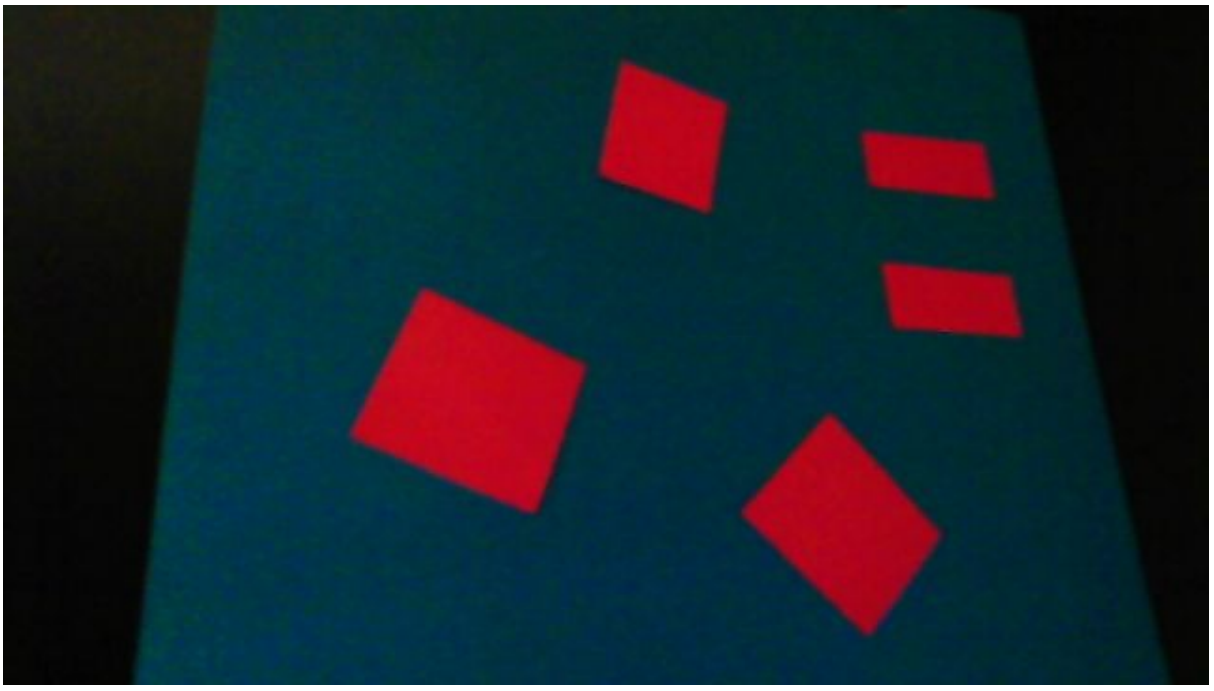
1. Funkcja dostarczająca obraz

```
public void TakePhoto()
{
      Texture2D photo = new Texture2D(webCamTexture.width,
      webCamTexture.height);
      photo.SetPixels(webCamTexture.GetPixels());
      photo.Apply();


      byte[] bytes = photo.EncodeToPNG();
      File.WriteAllBytes(Application.dataPath +
      "/CameraPictures/CameraInput.png", bytes);
}
```

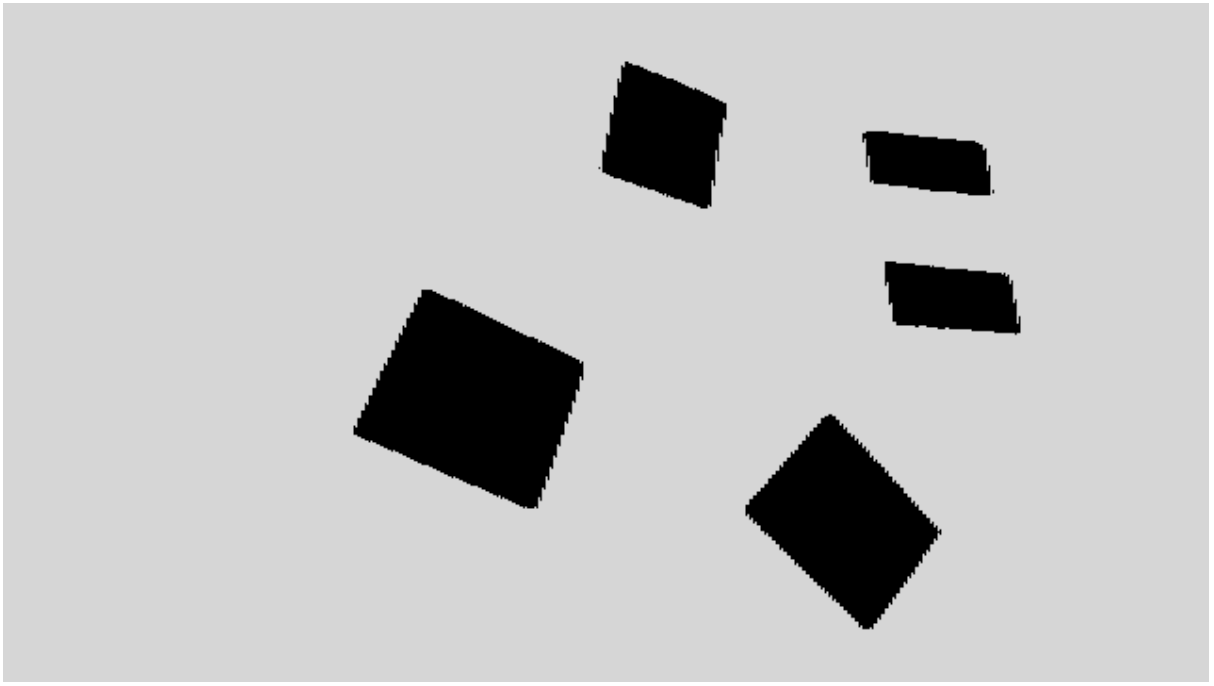
2. Wykrywanie barw

```
if (File.Exists(Application.dataPath + imagePath))
{
      fileData = File.ReadAllBytes(Application.dataPath +
      imagePath);
      image = new Texture2D(2, 2);
      image.LoadImage(fileData);

      Texture2D newTex = new Texture2D(image.width,
      image.height);


      for (int x = 0; x < newTex.width; x++)
      {
            for (int y = 0; y < newTex.height; y++)
            {

                  var rgb = image.GetPixel(x, y);
                  float H, S, V;
                  Color.RGBToHSV(rgb, out H, out S, out V);

                  if (H > HvalueMin && H < HvalueMax && S <
                  SvalueMax && S> SvalueMin && V > VvalueMin && V
                  < VvalueMax )
                  {
```

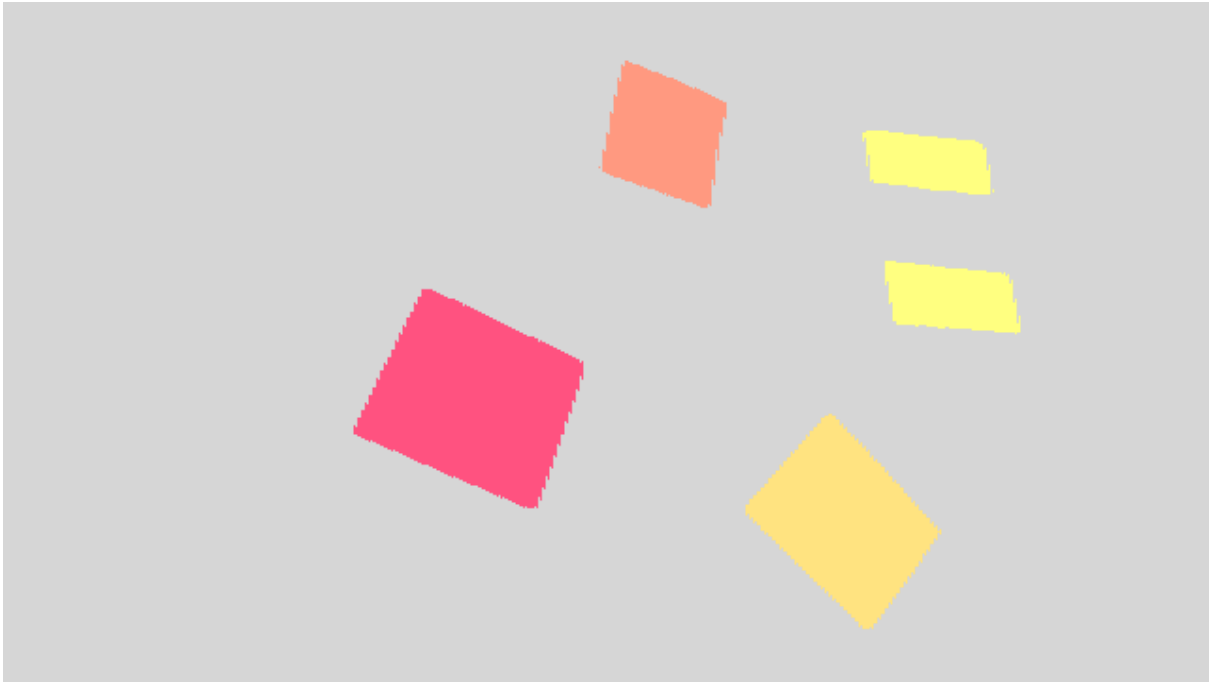
```
                    mapa[x, y] = 1;
                }
                    else
                    {
                    mapa[x, y] = 0;
                    }

                }
        }
```
(...)



Zdjęcie oryginalne



Czarno-biała mapa. W programie wartości te przechowywane są w formie listy z wartościami 0 lub 1

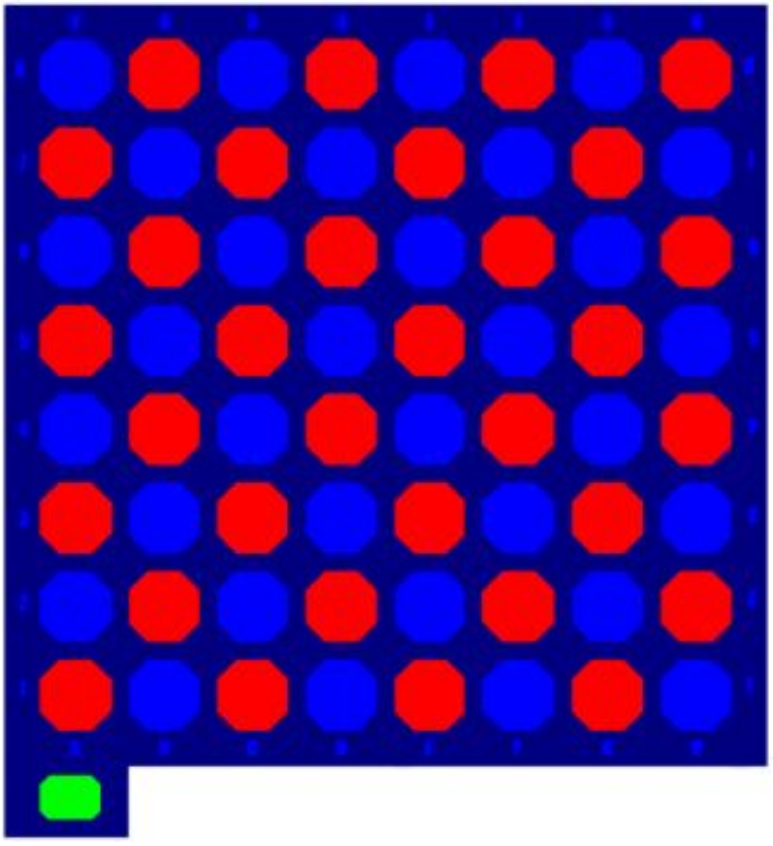


Zgrupowane obiekty. 5 różnych grup(2 grupy oznaczono podobnymi kolorami) przechowywane w listach z wartościami typu int rozpoczynając od 2

Takie działania przeprowadzamy dla każdego z 5 kolorów, które domyślnie oznaczają odpowiednio:

- czarny i biały -pionki graczy,
- czerwony -pola na których ustawione są pionki,
- zielony -jest to pojedyncze pole na planszy umiejscowione pod polem A1. Służy do identyfikacji tego właśnie pola, dzięki czemu planszę można dowolnie obracać względem kamery nie wywołując błędów w wizualizacji.
- szary -pionek damka

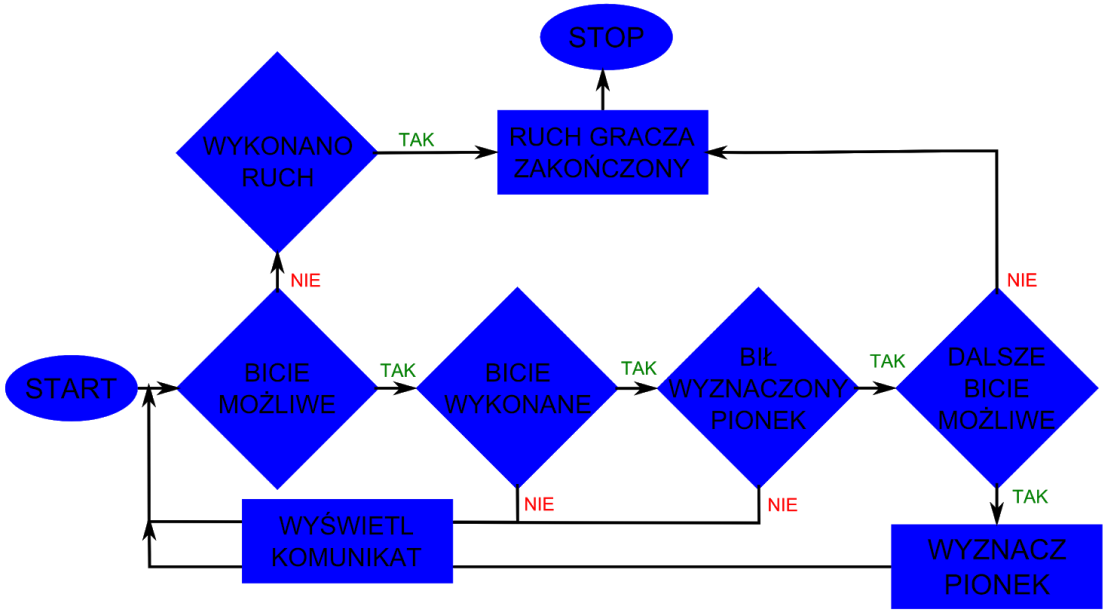
Kolory są umowne i można je zmieniać za pomocą suwaków wartości HSV. Jeśli np. ustawimy wartości koloru żółtego dla pól, to jako pola będziemy wykrywać kolor żółty na planszy i analogicznie wszystkie inne obiekty.
Dla przyjętych przez nas kolorów zaprojektowaliśmy następującą planszę:



Pionki natomiast nie zostały w żaden sposób spreparowane- to zwykłe bierki do gry warcaby w kolorach białym i czarnym.

4.2 Architektura modułu walidującego

Moduł walidacyjny to skrypt nazwany “Game Master”. Zawiera on wszystkie reguły gry i pilnuje ich przestrzegania oraz przechowuje informację o graczach (liczba pionków, wykonany ruch, tura itd). Został on zbudowany na podstawie następującego schematu blokowego:



Jest to uproszczona wersja całego modułu ukazująca jedynie jego fundamentalne zasady. W kodzie skryptu występują dodatkowe ograniczenia wynikające z faktu iż informację przekazywane do sstemu przez kamerę pochodzą z realnej planszy, na którą wpływ mają gracze. Niesie to możliwość próby oszustwa lub zniekształcenia obrazu w wyniku czego pionki mogą znikać i pojawiać się w niepoprawny sposób. Z tego powodu nasz “Mistrz gry” sprawdza dodatkowo następujące zasady:

1. Gracz wykonujący ruch nie może ani zmniejszać, ani zwiększać sumy swoich pionków.
2. Gracz nie wykonujący ruch nie może tracić pionków jeśli nie zostało wykonane bicie.
3. Gracz nie wykonujący ruch musi tracić ilość pionków równą ilości zbitych w turze pionków.
4. Gracz wykonujący ruch może przemieścić tylko 1 pionek własny.
5. Gracz nie wykonujący ruch nie może przemieszczać swoich pionków, może jedynie stracić je poprzez zbicie.

Kolejną rzeczą jest sprawdzenie poprawności obu ruchów - zarówno bicia jak i przemieszczenia pionka. W tym celu stworzyliśmy 2 biblioteki - funkcję zawierające zbiór zasad dla każdego pola:

- funkcja SąsiedziPola(klasa Pole) - funkcja przyjmuje klasę polę i wypełnia listę “sąsiedzi” tego pola odpowiednio zdefiniowanymi sąsiadami
- funkcja PolaBicia(klasa Pole) - funkcja przyjmująca klasę pole i wypełniające listę “bicia” tego pola odpowiednio zdefiniowanymi polami

Obie funkcję są wywoływane dla każdego pola podczas tworzenia planszy. Następnie po każdym ruchu wyznaczamy pole, z którego przemieścił się pionek w oparciu o porównanie dwóch list z zajętymi polami oraz sprawdzamy czy nowe pole, które zajmuje pionek jest odpowiednim sąsiadem tego, które zajmował w turze poprzedniej. Jeśli otrzymamy wartości true jako wynik przemieszczenia ruch zostaje zatwierdzony, jeśli false na ekranie pojawia się komunikat o błędnym ruchu.

4.3 Architektura modułu wizualizacji

Po uruchomieniu programu wyświetlane jest menu główne z następującymi opcjami:

1. Przejście do ekranu głównego wizualizacji.
2. Wyświetlenie zasad gry.
3. Zakończenie działania aplikacji.

Po wciśnięciu przycisku “Start” na ekranie wyświetli się okno z pełnym interfejsem użytkownika, na który składa się scena utworzona w Unity, przechwytywany obraz z kamery oraz funkcjonalne przyciski, czy też etykiety informacyjne. Szczegółowo elementy te zostały opisane poniżej:

Zrzut ekranu z wizualizacji rozgrywki wraz z zaznaczonymi elementami, które zostały opisane poniżej.

W sekcji oznaczonej numerem 1 użytkownik ma dostęp do ustawień i podglądu przechwytywanego obrazu. Może tutaj ustawić przedziały kolorów dla poszczególnych obiektów, klatki na sekundę, odstępy między obiektami oraz przeprowadzić testy

W sekcji numer 2 znajduje się obraz odbierany przez kamerę.

Sekcja numer 3 to wizualizacja planszy na której pojawiają się odpowiednie pionki podczas rozgrywki.

W sekcji numer 4 znajdują się funkcjonalne przyciski, z których każdy realizuje odpowiednie zadanie:

- Przycisk Play - odczytuje pozycje pionków, nanosi je na planszę oraz kontroluje ich położenie w oparciu o zaimplementowane reguły
  To tutaj możemy też za pomocą obiektu toogle wybrać czy po rozpoczęciu gry pierwszy ruch wykonają pionki czarne czy białe

- Detect Board wykrywa planszę z przechwytywanego przez kamerę obrazu. Skrypt ukryty pod tym przyciskiem wykonuje migawkę obrazu, zczytuje kolory, zlicza liczbę pól w określonym kolorze i przypisuje je do obiektów widocznych na scenie. Nad przyciskiem znajduje się informacja o tym czy plansza została wykryta

- WebCamera Switch pozwala na przełączanie się między kamerami internetowymi, jeśli użytkownik ma podłączoną więcej niż jedną kamerę do jednostki.

Sekcja numer 5 wyświetla podstawowe informacje o rozgrywce tj.:
- tura - kolejka, w której gracze wykonują ruchy
- ilość pionków czarnych i białych
- gracz aktualnie wykonujący ruch

W sekcji oznaczonej numerem 6 możemy zobaczyć aktualny status rozgrywki, ostatni wykonany ruch oraz komunikat o tym czego oczekuje w danej chwili aplikacja

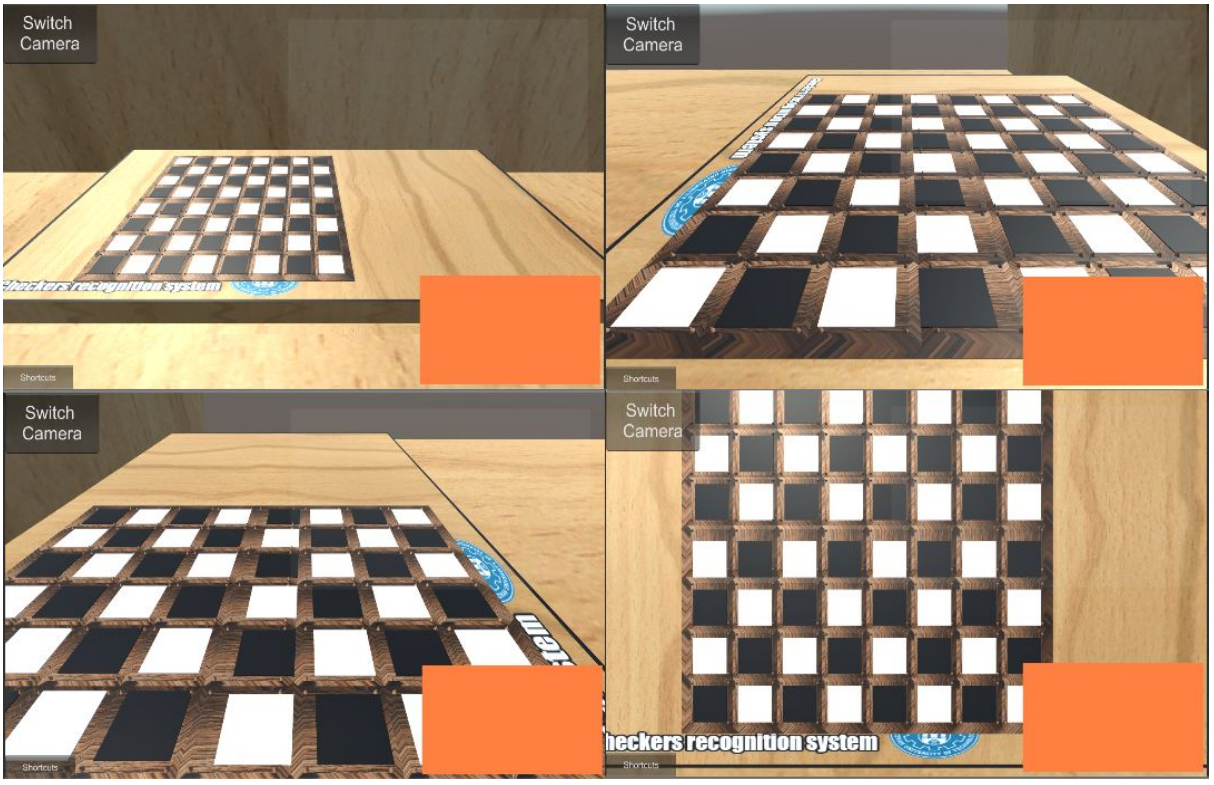
4.4 Opis dodatkowych funkcjonalności

W aplikacji zostały dodane dodatkowe opcje, niebędące wymaganiami funkcjonalnymi, lecz zwiększające komfort użytkowania aplikacji, a także ułatwić pewne operacje związane z rozgrywką.

Przełączanie widoku

Za pomocą dodatkowych skryptów umożliwiliśmy użytkownikowi zmianę perspektywy. Gracz może przełączać się między widokami z czterech różnych

kamer: frontowej, dwóch kamer bocznych oraz widok z lotu ptaka. Przełączanie widoku można wykonać wciskając klawisz ‘C’.



Zrzuty ekranu z rozgrywki prezentujące dostępne widoki
(Zrzuty zostały wykonane podczas fazy projektowania sceny w Unity, nie ukazują zaimplementowanego GUI)

Restart sceny

Wśród skryptów umieściliśmy również opcję restartu sceny, która przywraca stan wizualizacji rozgrywki do postaci startowej - kamery frontowej.

Zmiana kamery internetowej

Przycisk “Switch webcam” umożliwia zmianę kamery, z której przechwytujemy obraz. Wszystkie kamery wykryte w systemie są przechowywane w tablicy jednowymiarowej, po której poruszamy się poprzez naciśnięcie przycisku.

Przycisk Play z możliwością wyboru gracz

Wznawianie rozgrywki było bardzo kłopotliwe w przypadku zakończenia ruchu w kolejce gracza czarnego, jako że gra rozpoczynają domyślnie pionki białe. Z tego powodu dodaliśmy przycisk toogle, który umożliwia zmianę gracza rozpoczynającego grę. W ten sposób nie tylko możemy kontynuować grę przerwaną podczas ruchu pionków czarnych, ale możemy też decydować jaki kolor ma rozpoczynać grę.

5. Instrukcja użytkowania aplikacji

Instrukcja obsługi jest najważniejszym źródłem informacji o prawidłowym działaniu systemu, dlatego podstawą do tego, aby użytkownik mógł korzystać z aplikacji konieczne jest dostarczenie mu dokładnego opisu obsługi programu. Podczas pierwszego kontaktu z aplikacją ważne jest, by użytkownik wiedział, jakie czynności musi wykonać, ale również by rozumiał, dlaczego je wykonuje oraz jaki mają one wpływ na działanie programu.

5.1 Wymagania systemowe

Do przeprowadzenia rozgrywki z wykorzystaniem systemu użytkownik musi zaopatrzyć stanowisko w następujące elementy:

- Kamera internetowa lub smartfon z aplikacją DroidCam.
- Plansza do gry w Warcaby klasyczne (8 x 8) wraz ze znacznikiem w innym kolorze oraz pionki w 2 różnych kolorach z oznaczeniem damek w innym kolorze. Pionki powinny mieć mniejszą średnicę niż długość boku pojedynczego pola planszy.
- System operacyjny Windows 10 lub Windows 7(Projekt był uruchamiany na tych systemach)
- Monitor o rozdzielczości co najmniej 1280 x 728.
- Minimum 512 MB wolnej pamięci operacyjnej.
- W celu zachowania płynności przetwarzania obrazu wymagany jest dwurdzeniowy, wielowątkowy procesor o taktowaniu minimum 2 GHz.

5.2 Pełna instrukcja obsługi systemu

Przed uruchomieniem aplikacji wymagane jest przygotowanie pomieszczenia do gry, w celu poprawnego przetwarzania obrazu istotne jest światło w jakim rozgrywka będzie się odbywać. Optymalnym rozwiązaniem w tym przypadku będzie światło białe, czyli takie, które najbardziej przypomina światło dzienne. Barwa tego światła mieści się w okolicach temperatury 4000 Kelwinów. Warto unikać bezpośredniego kierowania światła na planszy - zależnie od materiału, z którego jest ona wykonana mogą pojawić się odbicia, a sama plansza może zostać nierównomiernie oświetlona, co z kolei prowadzić może do błędnego odbioru obrazu przez system. Najlepszym rozwiązaniem jest zastosowanie silnego źródła światła i rozproszenie promieni świetlnych w pomieszczeniu. Wszelkie cienie czy inne problemy ze światłem mogą wpływać na niepoprawne działanie programu.

W kolejnym etapie należy przygotować stanowisko, czyli położyć planszę na płaskiej powierzchni, ustawić na niej pionki w układzie zgodnym z zasadami gry, następnie umieścić kamerę internetową nad planszą, tak aby plansza, pionki oraz ‘marker’ określający pozycję mieściły się w kadrze.
Dalej należy uruchomić system, a z interfejsu graficznego kliknąć przycisk ‘Detect Board’, który podejmie próbę wykrycia planszy. O tym czy zadanie to powiodło się czy nie poinformuje nas komunikat nad przyciskiem. W przypadku niepoprawnego wykrycia planszy uruchomienie rozgrywki nie będzie możliwe.
Jeśli proces przygotowania rozgrywki przebiegł poprawnie, a system wykrył planszę oraz pionki, to można przejść do rozpoczęcia gry - poprzez wciśnięcie przycisku ‘Play’. Program wykryje wtedy położenie pionków oraz naniesie je na planszę. Następnie stosowne komunikaty wyświetlać się będą na pasku komunikatów. Należy stosować się do owych instrukcji oraz przestrzegać zasad gry.

6. Podsumowanie

Głównym założeniem projektu, było utworzenie systemu, który umożliwia cyfrową wizualizację rozgrywki w Warcaby.
Pomimo napotkanych problemów i zmiany technologii, które nastąpiły w trakcie pracy, nasz zespół zrealizował przyjęte cele.

6.1 Napotkane problemy i ich rozwiązania

Z każdą pisaną aplikacją wiąże się seria problemów, na które programista/zespół natrafia w trakcie pracy i bardzo często ich rozwiązanie wymaga znacznie więcej czasu niż napisanie kodu. W trakcie pracy nad projektem nasz zespół również nie uniknął komplikacji w tworzeniu systemu.

Początkowo nasz zespół zakładał wykorzystanie języka programowania C++ oraz biblioteki OpenCV do przechwytywania i przetwarzania obrazu z kamery z tą koncepcją wiązały się następujące problemy:

Problem 0. Brak Kamerki

Ze względu na to, że nie wszyscy z nas mieli kamerki internetowej musieliśmy poradzić sobie bez specjalistycznego sprzętu. Każdy z nas miał smartfon wyposażony w kamerę z systemem operacyjnym Android, dzięki czemu mogliśmy za pomocą aplikacji DroidCam zamienić telefon w kamerkę. Aplikacja łączy się z lokalną siecią bezprzewodową, a zainstalowane na komputerze oprogramowanie klienckie, pozwala przechwycić obraz kamery z telefonu. Z telefonem łączymy się poprzez adres IP podany w aplikacji na smartfonie;

Problem 1. Instalacja bibliotek openCV

OpenCV to zbiór bibliotek tak rozległych że do wygodnego korzystania należało dodać je do zmiennych środowiskowych w systemie Windows oraz do właściwości projektu w programie Visual Studio (sekcje C/C++ >> General oraz Linker >> General Input). Poprawna konfiguracja zajęła trochę czasu, jako że nie było to działanie standardowo wykonywane przy prostych projektach. Rozwiązaniem okazały się filmy instruktażowe innych użytkowników, które dokładnie opisały cały proces.

Problem 2. Śledzenie obiektów według barw

Śledzenie obiektów o różnych kształtach i kolorach nie jest łatwe bez zastosowania odpowiednich sposobów, dlatego zdecydowaliśmy się na osiągnięcie tego celu poprzez wykorzystania modelu HSV(hue, saturation, value) odcień, nasycenie i wartość światła by wyodrębnić obiekty pewnego koloru i śledzić je bez zlewania ich z tłem. Problem ten powrócił gdy w środowisku Unity próbowaliśmy wykrywać barwy po modelu RGB. W ostatecznej wersji zastosowaliśmy jednak model HSV tak samo jak przy pierwotnych próbach z biblioteką openCV. Rezultaty były znacznie lepsze.

Problem 3. Wykrywanie kamery w opencv

OpenCV domyślnie identyfikuje kamerę wbudowaną jako tą z indeksem 0. W przypadku dodatkowych kamer usb, sprawa komplikuje się i wielu użytkowników ma problemy z wykryciem prawidłowego id pożądanej kamerki. Rozwiązanie tego problemu okazało się dość trywialne. W pętli for spróbowaliśmy przechwytywać obraz z urządzeń z rożnych id z zaimplementowaną obsługą wyjątków. Dla 999 prób powiodły się tylko 3. 0-domyślna kamerka wbudowana, 700-domyślna kamerka wbudowana oraz 701- kamerka usb. Nie jest to rozwiązanie bardzo wyrafinowane, lecz skuteczne biorąc pod uwagę że indeks 701 trudno byłoby osiągnąć przy ręcznej próbie wylosowania odpowiedniego numeru id.

Problem 4. Graficzna wizualizacja stanu rozgrywki

Działając w C++ byliśmy w stanie bez problemu zwizualizować stan planszy w postaci tekstowej na aplikacji konsolowej, z doborem odpowiedniej biblioteki do utworzenia interfejsu graficznego, który cieszyłby oko wiązało się jednak więcej problemów. Na tym etapie zwątpiliśmy w początkową koncepcję i zaczęliśmy zastanawiać się nad zmianą narzędzi, które wykorzystamy do wykonania zadania.

Powyższe problemy skłoniły nas do zmiany wykorzystywanych technologii(C++ w połączeniu z OpenCV), po przeanalizowaniu możliwości i rozpoznaniu w podobnych projektach uznaliśmy, że najlepszym rozwiązaniem będzie wykorzystanie języka C#, w środowisku Unity, które dostarcza nam wszystkie narzędzia, których potrzebujemy do wykonania zadania - nie ma potrzeby korzystania z zewnętrznych bibliotek.

Problem 5. Synchronizacja postępów w Unity - GIT

Tworzenie tak złożonego systemu wymaga korzystania z systemu kontroli wersji, tak aby wszyscy członkowie zespołu mieli dostęp do najnowszej wersji aplikacji i mogli pracować jednocześnie nad projektem. W tym celu użyliśmy GITa. Podczas pracy w języku C++, w programie Visual Studio nie mieliśmy żadnych problemów z synchronizacją.
Komplikacje zaczęły pojawiać się po zmianie środowiska na Unity, które z nieznanych nam początkowo przyczyn spłatało nam wiele figli np. odłączało skrypty od obiektów, występowały problemy z importowaniem tzw. "Assets", silnik w nieskończoność kompilował skrypty, po kliknięciu w inne okno działające w systemie Windows i ponownym kliknięciu w okno Unity program na pewien czas zatrzymywał działanie, ponownie próbując importować obiekty, skrypty oraz inne komponenty, co wzbudzało irytację wśród członków zespołu. Pomimo starań od samego początku, by problemów z synchronizacją było jak najmniej (korzystanie z tej samej wersji Unity, nie importowanie customowych Assetów) opisane problemy występowały. Na szczęście na większość z nich zdołaliśmy znaleźć rozwiązanie, głównie polegające na braku synchronizacji pliku manifestu, zawierającego informacje o dodatkowych pakietach, który to trzeba było ręcznie zmieniać na manifest z własnego projektu utworzonego na maszynie, na której pracowaliśmy z projektem, ale wymagało to czasu.

Problem 6. Rozpoznawanie obrazu w Unity
Środowisko Unity ne dostarcza wbudowanych bibliotek do przetwarzania obrazu. Problem ten postanowiliśmy rozwiązać przez stworzenie własnych mechanizmów, które szczegółowo opisane zostały w rozdziale architektura.

Problem 7. Wykrywanie i sprawdzanie legalności ruchu
Bardzo szybko i bezproblemowo rozpoczęliśmy tworzenie funkcji sprawdzających czy pionek wykonał poprawny ruch, czy miejsce miało bicie, ale połączenie ich wszystkich w jedna spójną całość okazało się trudne ze względu na liczne “dziury” i nieprzewidziane przypadki ruchu gracza. Rozwiązaniem okazało się stworzenie

jednego wspólnego dla wszystkich ruchów schematu blokowego, który uwzględniał sposób sprawdzania ruchów, bez szczegółowego sprawdzania w jaki sposób mają być one wykonywane. Implementacja okazała się nadzwyczaj prosta, jako że pojedyncze funkcję były już stworzone i wystarczyło odpowiednio je uporządkować.

6.2 Realizacja założonych celów

Wszystkie założenia projektowe zostały zrealizowane. System poprawnie przetwarza obraz z kamery, poprawnie wyświetla położenie i rodzaj figur na planszy do gry oraz właściwie interpretuje wykonane ruchy. Nie wszystko zostało zrealizowane jednak idealnie- wykrywanie kolorów wymaga kalibracji dłuższej niż konieczne, a komunikacja systemu z użytkownikiem jest w kilku przypadkach ograniczona bardziej niż to konieczne ze względu na błędy przetwarzania obrazu. Ostatecznie jednak pomimo kilku niedogodności w użytkowaniu, kluczowe funkcję działają poprawnie, co pozwala uznać projekt za zrealizowany z powodzeniem.

6.3 Wnioski

Podstawowy wniosek, który nasuwa się po wykonaniu projektu dotyczy przetwarzania obrazu. Zdecydowaliśmy się nie korzystać z gotowych bibliotek, lecz stworzyć własne i mimo że zrealizowaliśmy tą decyzję to otrzymaliśmy słabsze wyniki oraz poświęciliśmy na ten cel za dużo czasu, którego brak odczuliśmy w końcowej fazie projektu.
Kolejną sprawą jest wykrywanie planszy oraz obiektów na niej z uwzględnieniem kolorów. Na własnej skórze, przy każdorazowej zmianie oświetlenia stanowiska odczuliśmy jak wadliwym było to rozwiązanie. Wyodrębnienie jednego koloru z obrazu stanowi umiarkowane zadanie, lecz jednoczesne wyodrębnienie 5 dostarczyło nam wiele nerwów, jako że zakresy barw musiałby być dokładnie sprecyzowane, tak by jedne obiekty nie były mylone z innymi.
Pomimo prób ulepszania wybranego przez nas rozwiązania (zmiana modelu RGB na HSV, traktowanie obiektów poza planszą jako cienie i odbicia, itp.) okazało się że nie możemy uzyskać dużej poprawny ponieważ wada leżała w wybranym rozwiązaniu, a nie jego implementacji.
Implementacja zasad zakończyła się powodzeniem - nie ma pionka, który nie byłby obsługiwany poprawnie, lecz przy projektowaniu algorytmów przestrzegania zasad, większy nacisk powinniśmy położyć na miejsca, w których system rozróżnia i komunikuje dokładne przyczyny niepoprawnego ruchu i błędy wykonywane przez użytkownika. Nie wystarczające jest by przestrzegać zasad, należy także dokładnie i szczegółowo przedstawić je użytkownikowi za każdym razem gdy ten popełnia błąd.
Dużym sukcesem zakończył się natomiast proces wizualizacji rozgrywki. Poznaliśmy wiele dotychczas nieznanych nam narzędzi w środowisku unity. Stworzyliśmy własne autorskie modele i odpowiednio zaprezentowaliśmy je na scenie w zależności od ruchów użytkownika. Czujemy wręcz niedosyt że nie zdążyliśmy dodać animacji, efektów dźwiękowych i innych dodatków audio-wizualnych, które wzbogaciłby wrażenia z rozgrywki.

6.4 Perspektywa rozwoju aplikacji

Każdy system, nawet jeśli spełnia założone wymagania i nie posiada błędów można zoptymalizować, bądź też rozbudować o dodatkowe funkcjonalności. W ramach naszych przemyśleń na temat zagadnień związanych z projektem w naszych głowach narodziły się następujące pomysły na rozwój aplikacji:

Pomysł 1. Odtworzenie gry

Jednym z pomysłów na jakie wpadliśmy to odtworzenie przebiegu rozgrywki z logu pojedynku. Log ten zawierałby informację o wszystkich wykonanych ruchach pionków, biciach i być może nawet złych posunięć graczy (osądzonych przez naszego “skryptowego” sędziego). Implementując taki dziennik w program, wprowadzenie możliwości odtworzenia meczu wydaje się być dość prosta. Dodatkowo moglibyśmy wprowadzić możliwość cofania wykonanego ruchu w oparciu o wyżej wspomniany log.

Pomysł 2. Kalibracja wykrywanych kolorów

Powodzenie wykrywanych obiektów na planszy zależy w bardzo dużym stopniu od oświetlenia panującego w pokoju. W świetle dziennym problemy z detekcją właściwie nie występowały, lecz po zachodzie słońca problem stał się bardzo duży. Niejednokrotnie traciliśmy sporo czasu na dopasowanie wartości RGB i w późniejszej fazie projektu HSV. Ułatwieniem byłaby kalibracja odczytywanego koloru. Poprzez stosowną opcję w menu, użytkownik byłby proszony o podsunięcie zadanego koloru pod obiektyw kamery, a program automatycznie przydzieliłby odpowiedni zakres szukanych barw.

Pomysł 3. Wprowadzenie animacji

W obecnej fazie nasz program, po wykryciu zmian na planszy zwyczajnie chowa pionki, z miejsca gdzie ma ich nie być i wstawia je w miejsce gdzie mają się znaleźć. Choć jest to jak najbardziej wystarczające do poprawnej reprezentacji stanu planszy i bardzo często wykorzystywane przez programistów tworzących gry, to mimo wszystko jest to mało eleganckie rozwiązanie. Zdecydowanie milej oglądało by się przebieg rozgrywki na monitorze komputera, gdybyśmy dodali mu animacje.
Tym bardziej, że Unity jest wręcz stworzone do tego celu i w internecie jest mnóstwo materiałów na ten temat. Moglibyśmy dodać animację na początku gry (np. plansza spadająca z góry na stół), zbity pionek mógłby płonąc chwilę po czym zniknąć, zamiana zwykłego pionka w damę, czy fajerwerki na końcu pojedynku.

Pomysł 4. Spiker ogłaszający wykonanie ruchu

Jednym z najczęstszych błędów użytkownika jest wykonanie ruchu zanim program zdążył odczytać ruch poprzedni. By zmniejszyć ryzyko błędu można by dodać spikera lub dźwięk informujący o wykryciu ruchu, bez którego przeciwnik wiedziałby że nie powinien przesuwać swoich pionków