2012

Stereogramiksy

Projekt z Grafiki Komputerowej

Michał Golonka - Michał Franczyk - Szymon Jagieła
WFiIS kierunek Informatyka Stosowana
2012-02-06

I. Stereogramiksy.

1. Wstęp.

Tekst stanowi dokumentację programu o nazwie „Stereogramiksy” i zawiera wszelkie informacje odnośnie projektu, jego organizacji, konstrukcji, jak również wykorzystywanego środowiska programistycznego, użytych bibliotek oraz algorytmów, a także pełny opis działania w kontekście postawionych wymagań, sposób komunikacji z użytkownikiem i operacje, których wykonanie umożliwia wraz z określeniem formatu danych, na jakich operacje te będą wykonywane.

2. Autorzy projektu.

Zespół projektujący stanowią 3 osoby o wstępnie różnych rolach:

- **Michał Franczyk** – lider, określenie wymagań, wyszukiwanie potrzebnych informacji, implementacja głównego algorytmu, optymalizacja kodu, poprawki w GUI, tester

- **Szymon Jagieła** – pisanie kodu, tworzenie GUI, poszerzanie programu o nowe moduły, tester, rozplanowanie dokumentacji

- **Michał Golonka** – wyszukiwanie informacji, poprawki w kodzie oraz GUI, tester, pisanie dokumentacji programu

Funkcje opisane zostały ogólnikowo w ramach wstępu. Zakres działań każdego członka zespołu został szerzej omówiony w punkcie V, zawierającym opis podziału pracy, a także przybliżony czas poświęcony na realizację postawionych celów.

II. Opis projektu.

1. Geneza.

Projekt powstał w ramach przedmiotu „Podstawy Grafiki Komputerowej” i został wykonany przez studentów III roku (V semestru) informatyki stosowanej Wydziału Fizyki i Informatyki Stosowanej AGH. Przedmiot prowadzony jest przez dr hab. inż. Jacka Tarasiuka, będącego równocześnie pomysłodawcą rozważanego w projekcie zagadnienia, jak również doradcą i dostarczycielem niezbędnych informacji w procesie jego realizacji.

2. Rozpatrywany problem.

Tematem, jaki został poruszony w projekcie, są stereogramy. Przy rozpoczęciu procesu tworzenia konieczne było udzielenie odpowiedzi na kilka podstawowych pytań, jak np. czym są, jak powstają, a w dalszej kolejności, jak samodzielnie je wytworzyć.

Stereogram – inaczej nazywany obrazem stereoskopowym, jest obrazem w taki sposób przedstawianym na płaszczyźnie, aby przy użyciu odpowiedniej techniki patrzenia dawał złudzenie obrazu trójwymiarowego. Znalezienie „głębi” takiego obrazu jest możliwe po rozproszeniu wzroku, czyli skupieniu jego ogniskowej za płaszczyzną z obrazem. W tym celu można także posłużyć się odpowiednimi narzędziami, jakimi mogą być np. specjalne okulary o różnobarwnych szkiełkach. Istnieją również techniki tworzenia umożliwiające zobaczenie stereogramu przez skupienie wzroku przed płaszczyzną, czyli, potocznie rzecz ujmując, zrobienie zeza.

Powyższe zagadnienie znane było już w XIX wieku, a tworzone wtedy stereogramy składały się z dużej liczby losowo położonych, kolorowych kropek. Z czasem, ta technika była doskonalona, jednak ręcznie nadal możemy tworzyć tylko proste obrazy stereoskopowe.

III. Założenia wstępne.

1. Wymagania podstawowe.

Projekt ma spełniać następujące założenia:
- Program wczytuje bitmapę.
- Program przy użyciu odpowiednich procedur tworzy na podstawie wczytanej bitmapy z obrazem w skali szarości obraz stereoskopowy.
- Program zapisuje wygenerowany obraz jako bitmapę w rozmiarze 800x600.

2. Wymagania dodatkowe.

W ramach udoskonalenia projektu i poszerzenia o nowe funkcjonalności postawiono cele:
- Program wczytuje obraz nie tylko w formacie „bmp", jednak przy zaznaczeniu, że tylko te w skali szarości mogą zagwarantować poprawne działanie.
- Program może zapisywać bitmapy nie tylko w rozmiarze 800x600 pikseli.
- Program będzie również wyposażony w oddzielny moduł, czyli grę.
- Program zostanie także zaopatrzony w opcję tworzenia własnej gry.

Porównanie z podstawowymi wymaganiami:
Podane w dodatkowych wymaganiach założenia mogą robić wrażenie, jakby stały w sprzeczności z podstawowymi, jednak, analizując je pod kątem logiki, można zauważyć, że nie stanowią one w żaden sposób wykluczenia ani też ograniczenia pierwotnych celów. Są natomiast rozszerzeniem funkcjonalności projektu, dając mu możliwość „ogólniejszej" pracy oraz zapisu.

3. Ewolucja.

Wiele nowych pomysłów pojawiło się już w trakcie realizowania założeń, stąd też nie zostały one wymienione w powyższych. Dodatkowe opcje zostały wprowadzone na różnych etapach tworzenia, a najwięcej z nich w czasie implementacji. Wśród nich można wymienić: wybór rozmiaru, w jakim będzie zapisywany obrazek (w tym także zgodnie z założeniami 800x600), generowanie stereogramu w kolorze oraz wyrysowanie kropek, które mogą niektórym pomóc w zobaczeniu stereogramu.

IV. Analiza projektu.

1. Określenie danych wejściowych.

Jako dane wejściowe program przyjmuje obraz w formatach „bmp”, „jpg” oraz „png”. Obraz nie musi być w skali szarości, jednak jest to warunek konieczny do poprawnego działania. Program sam rozpoznaje, czy wczytany został obrazek w skali szarości, czy nie. Pozostałe parametry odpowiadające za końcowy rezultat pracy wybieramy sami w odpowiednim oknie programu przez zaznaczenie jednej z kilku gotowych opcji. Prawie każdy z tych parametrów jest liczbą rzeczywistą, a jego wartość w mniejszym lub większym stopniu warunkuje działanie zastosowanego algorytmu.

Dane wejściowe główne: plik graficzny *.bmp / *.jpg / *.png (najlepiej w skali szarości)
Dodatkowe parametry na wejściu: (ustalenie możliwe dopiero po wczytaniu obrazka)

- DPI ekranu – liczba naturalna

Ma znaczenie już w początkowych fazach działania algorytmu generującego stereogram, zwiększa jego elastyczność poprzez dodanie możliwości dopasowania się do atrybutów technicznych maszyny, na jakiej uruchomiony został program (domyślnie 96).

- Rozstaw oczu – liczba zmiennoprzecinkowa

Również ważne w początkowych fazach działania algorytmu, używany w tym samym punkcie, co parametr DPI, dodaje elastyczności przez możliwość dopasowania się do indywidualnych cech fizycznych użytkownika (domyślnie 2,7).

- Dopasowanie do rozmiaru okna – wartość logiczna

Określa, czy obrazek wraz ze zmianą rozmiaru okna programu zachowa swój współczynnik proporcji, czy też nie (domyślnie aktywny).

- Wygenerowanie pomocniczych kropek – wartość logiczna

Nie ingeruje w dużym stopniu w główny algorytm, rysuje duże kropki u dołu wczytanego obrazka w celu ułatwienia oglądania obrazu stereoskopowego (domyślnie nieaktywny).

- Odwrócenie wklęsłości – wartość logiczna

Wpływa na algorytm przez ustalenie sposobu tworzenia pozornej głębi, a więc, czy elementy ciemniejsze będą „bliżej”, czy „dalej” w stosunku do jaśniejszych (domyślnie nieaktywny).

- Kolor – liczba naturalna

Jest brany pod uwagę w końcowej fazie działania algorytmu, odpowiada za kolor wygenerowanego obrazka (domyślnie biały).

- Zapis w rozdzielczości – dwie liczby naturalne

Używany dopiero przy zapisie do pliku, wymusza na algorytmie ustalenie rozmiaru generowanego obrazka przed dokonaniem właściwych przekształceń (domyślnie 800x600).

- Keeping aspect ratio – wartość logiczna

Istotny dopiero przy zapisie do pliku, ustala, czy obrazek zostanie zapisany w swoich oryginalnych proporcjach, czy w wybranych przez użytkownika (domyślnie aktywny).

2. Oczekiwane dane wyjściowe.

Na wyjściu powinniśmy otrzymać obraz odpowiednio przekształcony przez wykorzystany algorytm. Przy zachowaniu domyślnych parametrów wejściowych ma on taki sam rozmiar, jak wczytywany obrazek i ma postać losowo rozsypanych, czarnych punktów na białym tle. Dodatkowo w oknie zostaje wyświetlonych kilka informacji na jego temat. Obrazek taki możemy powtórnie poddać działaniu algorytmu ze zmienionymi parametrami oraz zapisać do pliku graficznego w formacie „bmp", „jpg" lub „png", o wybranych rozmiarach.

Dane wyjściowe główne: obiekt graficzny, przy zapisie plik graficzny *.bmp / *.jpg / *.png
Dodatkowe informacje na wyjściu:

- Rozdzielczość – dwie liczby naturalne
Informuje, jaka jest szerokość i wysokość wczytanego obrazka (w pikselach).

- Rozmiar pliku – liczba zmiennoprzecinkowa
Podaje wielkość miejsca zajmowanego przez obrazek na dysku (w MB).

- Obrazek w skali szarości – wartość logiczna
Informuje, czy wczytany obrazek jest w skali szarości (tak/nie).

3. Struktury danych.

Większość danych używanych w programie ma postać pojedynczych zmiennych o pewnym przydzielonym obszarze w pamięci operacyjnej, aczkolwiek wykorzystane zostały również tablice zmiennych i listy. Niemal wszystkie są składnikami prywatnymi ogólniejszej struktury, a więc nie ma możliwości bezpośredniego odwoływania się do nich z zewnątrz. Są one natomiast ogólnodostępne wewnątrz i pracuje na nich wiele funkcji. Wywołanie odpowiednich procedur w celu nadania zmiennym pożądanej wartości jest możliwe tylko przy pomocy graficznego interfejsu użytkownika.

4. Interfejs użytkownika.

Służy komunikacji pomiędzy użytkownikiem a programem. Został wyposażony w odpowiednie opisy oraz obrazki odwołujące się do intuicji, dzięki czemu jest łatwy w obsłudze. Użytkownik ma wpływ na działanie programu za pomocą przycisków, list i formularzy.

GUI składa się z ośmiu okien:
1) Okno główne
2) Okno generatora
3) Okno wyboru gry
4) Okno gry I
5) Okno gry II
6) Okno kreatora scenariuszy I
7) Okno kreatora scenariuszy II
8) Okno z informacjami o programie

Okno główne:

To okno, które uruchamia się przy starcie aplikacji. Nie umożliwia zapoznania się z najważniejszymi funkcjami, a jest jedynie punktem wyjściowym w jej użytkowaniu i stanowi łącznik między oknami, w których funkcje te są już dostępne, a rezultaty ich działania zauważalne. Z tego poziomu użytkownik może jedynie uruchomić inne okno, będące graficznym reprezentantem określonego modułu programu, albo zamknąć program.

PRZYCISK: Generuj stereogram
Otwiera okno generatora stereogramów.

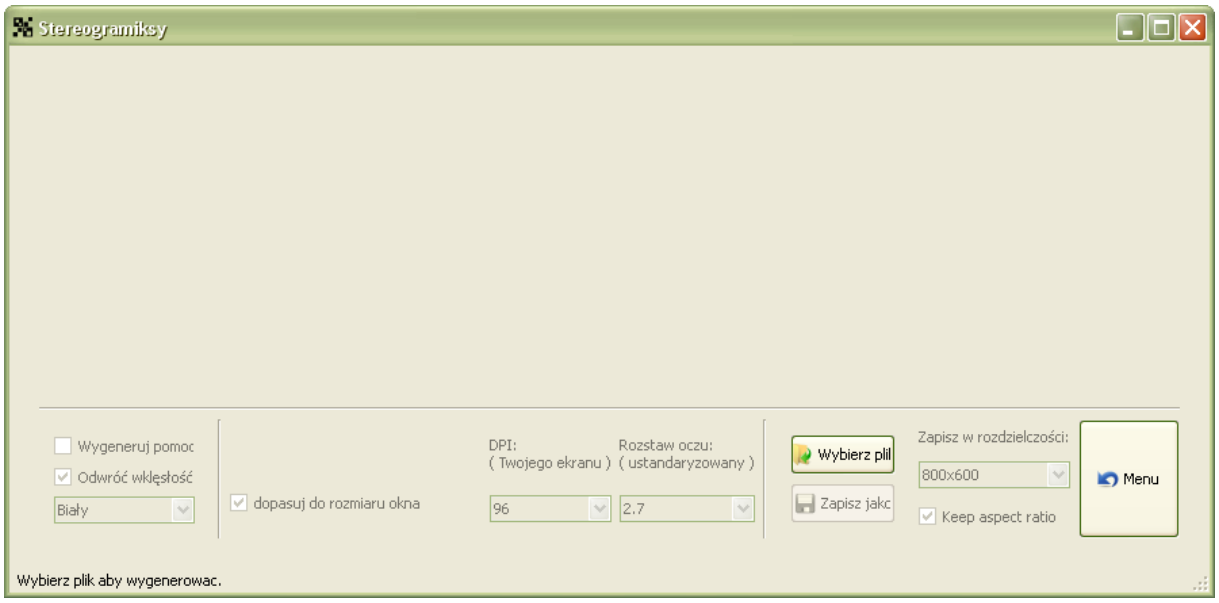
PRZYCISK: Graj
Uruchamia okno wyboru gry.

PRZYCISK: Stwórz scenariusz
Uruchamia okno kreatora scenariuszy I.

PRZYCISK: O programie
Otwiera okno z informacjami o programie.

PRZYCISK: Koniec
Zamyka okno główne i kończy pracę programu.

Okno generatora:



Okno generatora, czyli graficzny interfejs głównego modułu programu, jakim jest generator stereogramów. Tutaj użytkownik może ustalić wszystkie wcześniej opisane dane wejściowe i zobaczyć efekty pracy aplikacji.

PRZYCISK: Menu
Wraca do okna głównego.

PRZYCISK: Wybierz plik
Otwiera okno z wyborem pliku do wczytania.

PRZYCISK: Zapisz jako (dostępny dopiero po wczytaniu pliku)
Otwiera okno zapisu do pliku.

LISTA ROZWIJANA: Zapisz w rozdzielczości (dostępna dopiero po wczytaniu pliku)
Pozwala wybrać rozmiary, w jakich zapisany zostanie wygenerowany obrazek. Opcje:
- 800x600 (domyślna)
- 1024x768
- 1280x720
- 1920x1080

LISTA ROZWIJANA: Rozstaw oczu (dostępna dopiero po wczytaniu pliku)
Daje możliwość zmiany parametru, z jakiego korzysta algorytm na taki, który jest lepiej przystosowany do naszych potrzeb, bardziej nam odpowiada. W domyśle, powinna się tu znajdować odległość prawego oka użytkownika od lewego. Dostępne są możliwości:
- 2.5
- 2.6
- 2.7 (domyślny)
- 2.755

LISTA ROZWIJANA: DPI (Twojego komputera) (dostępna dopiero po wczytaniu pliku)
Daje możliwość zmiany parametru, z jakiego korzysta algorytm na taki, który jest lepiej przystosowany do naszych potrzeb, bardziej nam odpowiada. W domyśle, powinna się tu znajdować wartość parametru DPI ekranu komputera, na którym program został uruchomiony. Dostępne opcje to:
- 76
- 96 (domyślny)
- 150

LISTA ROZWIJANA: po lewej (dostępna dopiero po wczytaniu pliku)
Pozwala zmienić tło za porozrzucanymi czarnymi punktami na jeden z dostępnych kolorów. Można wybrać opcję wielu kolorów, jednak należy pamiętać, że może ona znacznie utrudnić zauważenie stereogramu. Możliwości to:
- Biały (domyślny)
- Szary
- Fioletowy
- Purpurowy
- Różowy
- Szkarłatny
- Czerwony
- Pomarańczowy
- Brązowy
- Żółty
- Pistacjowy
- Zielony
- Morski
- Błękitny
- Indigo
- Niebieski
- Wielokolorowy (w pionie)
- Wielokolorowy (w poziomie)

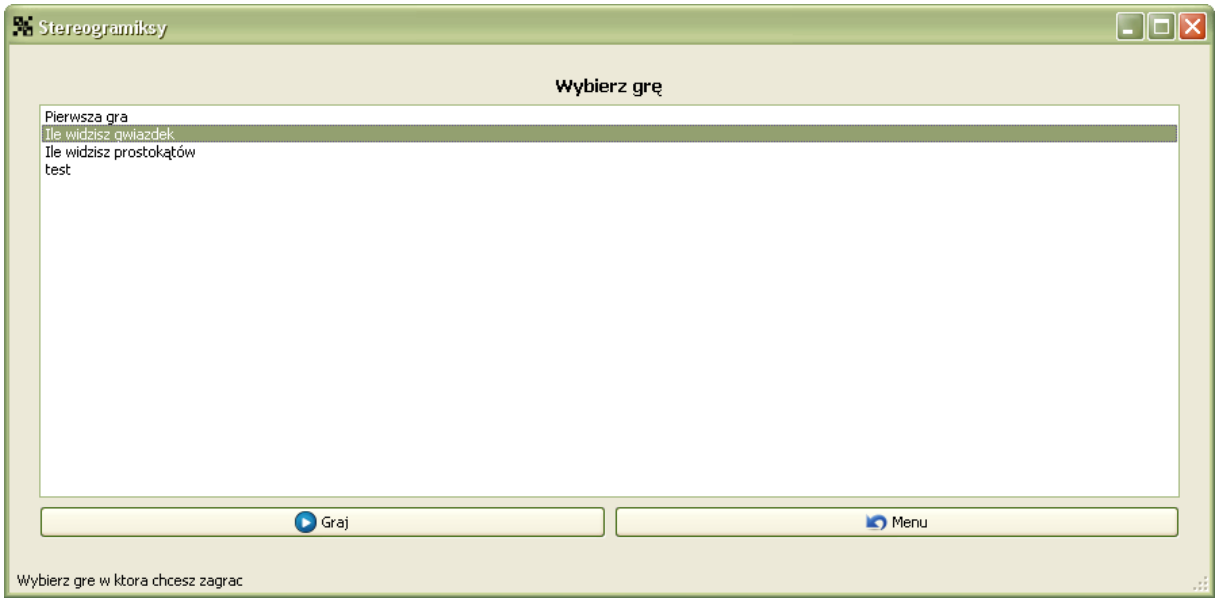
PUNKT: Keep aspect ratio (dostępny dopiero po wczytaniu pliku)
Wyłączony sprawia, że zapisywany do pliku obrazek będzie miał rozmiary ustalone przez użytkownika w odpowiedniej liście rozwijanej. Włączony pozwala na zapis w z oryginalnym współczynnikiem proporcji obrazka wykorzystanego jako źródło do wygenerowania stereogramu.

PUNKT: Dopasuj do rozmiaru okna (dostępny dopiero po wczytaniu pliku)
Włączony pozwala zachować wygenerowanemu obrazkowi na zachowanie współczynnika proporcjonalności w przypadku zmiany rozmiaru okna generatora. Wyłączony powoduje, że przy takiej zmianie obrazek traci swoje proporcje, dostosowując wysokość i szerokość osobno.

PUNKT: Wygeneruj pomocnicze kropki (dostępny dopiero po wczytaniu pliku)
Włączony rysuje u dołu wygenerowanego obrazka dwie czarne kropki. Wyłączony nie ma wpływu na obiekt obrazka oraz jego zachowanie.

PUNKT: Odwróć wklęsłość (dostępny dopiero po wczytaniu pliku)
Włączona powoduje, że stereogram tworzony jest z ustawieniem elementów ciemniejszych głębiej w stosunku do jaśniejszych, stąd robią one wrażenie elementów wklęsłych.
Wyłączony ustawia je bliżej niż części jaśniejsze, więc stają się one pozornymi wypukłościami.

Okno wyboru gry:



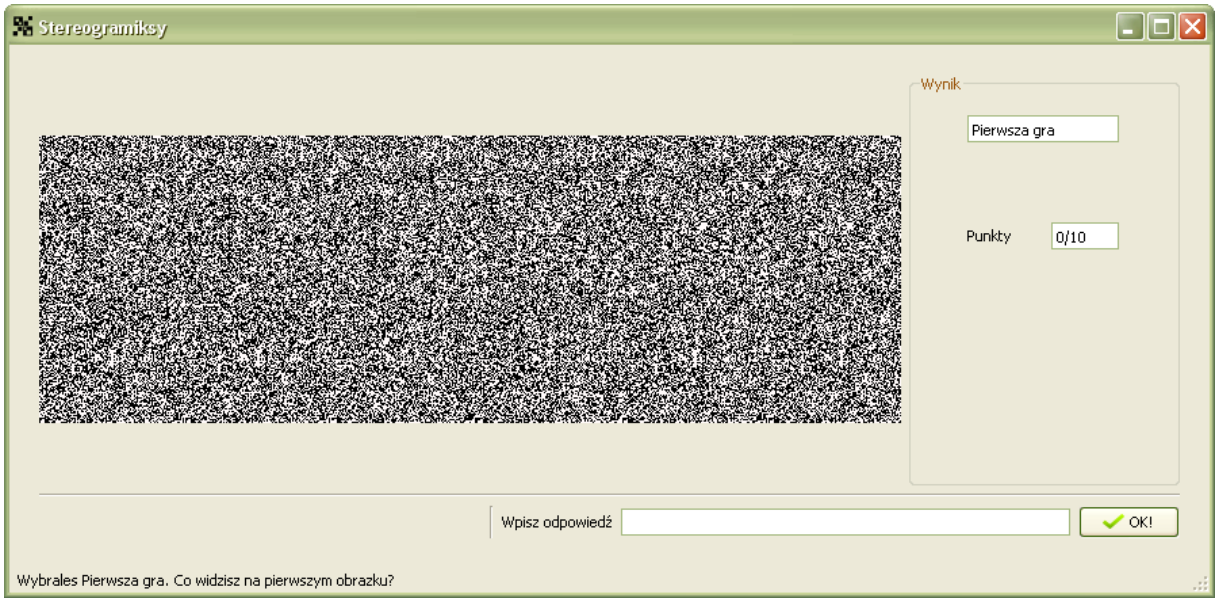
W oknie wyboru gry użytkownik ma możliwość uruchomienia gry, której tytuł zaznaczy na liście dostępnych. Może również wrócić do okna głównego programu, jeśli nie zdecyduje się zagrać w żadną.

LISTA: Wybierz grę
Przedstawia listę możliwych do wybrania gier, korzysta z odpowiedniego pliku tekstowego.

PRZYCISK: Graj
Uruchamia okno gry z tą, której tytuł został zaznaczony na liście.

PRZYCISK: Menu
Otwiera okno główne aplikacji.

Okno gry I:



Jest to okno, w którym wyświetlane są kolejne obrazki będące etapami gry. Powrót do okna głównego jest możliwy dopiero po przejściu do rezultatów, więc użytkownik, rezygnując z dalszej gry, może po prostu nie udzielać odpowiedzi i zatwierdzić wszystko za pomocą przycisku „OK!”.

PRZYCISK: OK!
Zatwierdza wyrażenie wprowadzone w polu edytowalnym. Ładuje kolejny obrazek lub, w przypadku, gdy bieżący etap gry jest ostatnim, otwiera okno z wynikiem, czyli okno gry II.

POLE EDYTOWALNE: Wpisz odpowiedź
To pole tekstowe, w którym użytkownik może wpisać dowolne wyrażenie, jednak tylko jedno jest poprawne i pozwala na zdobycie punktów za bieżący etap gry.

POLE TEKSTOWE: Punkty
Pole, w którym przedstawiony zostaje aktualny wynik gracza oraz maksymalna liczba punktów możliwych do zdobycia.

POLE TEKSTOWE: u góry
Pole z tytułem bieżącej gry.

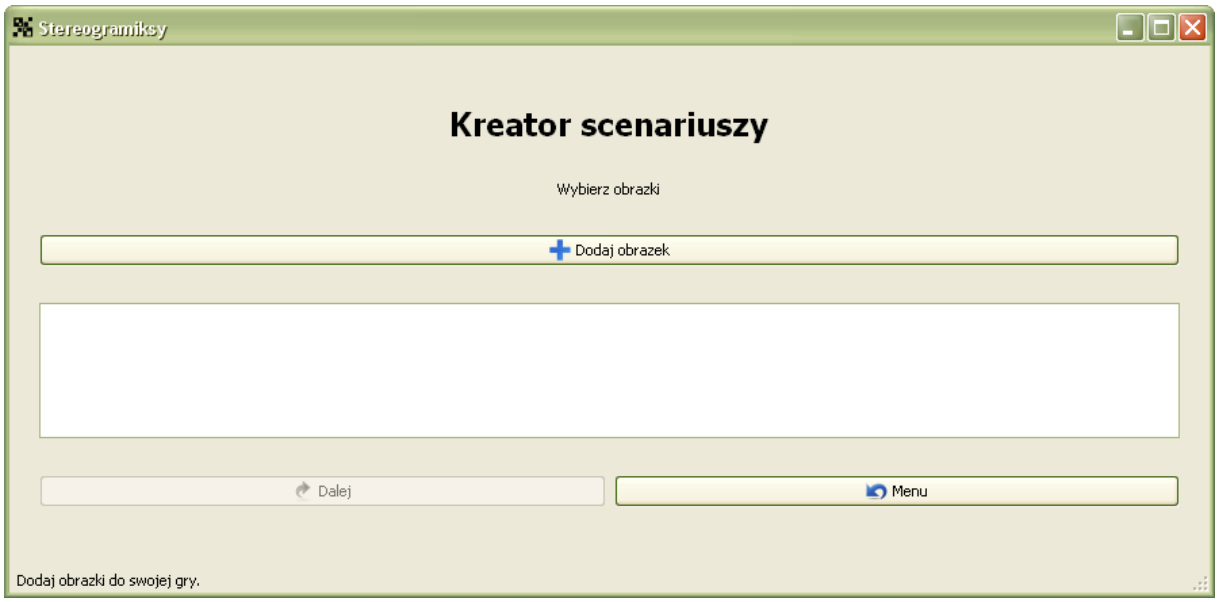
Okno gry II:

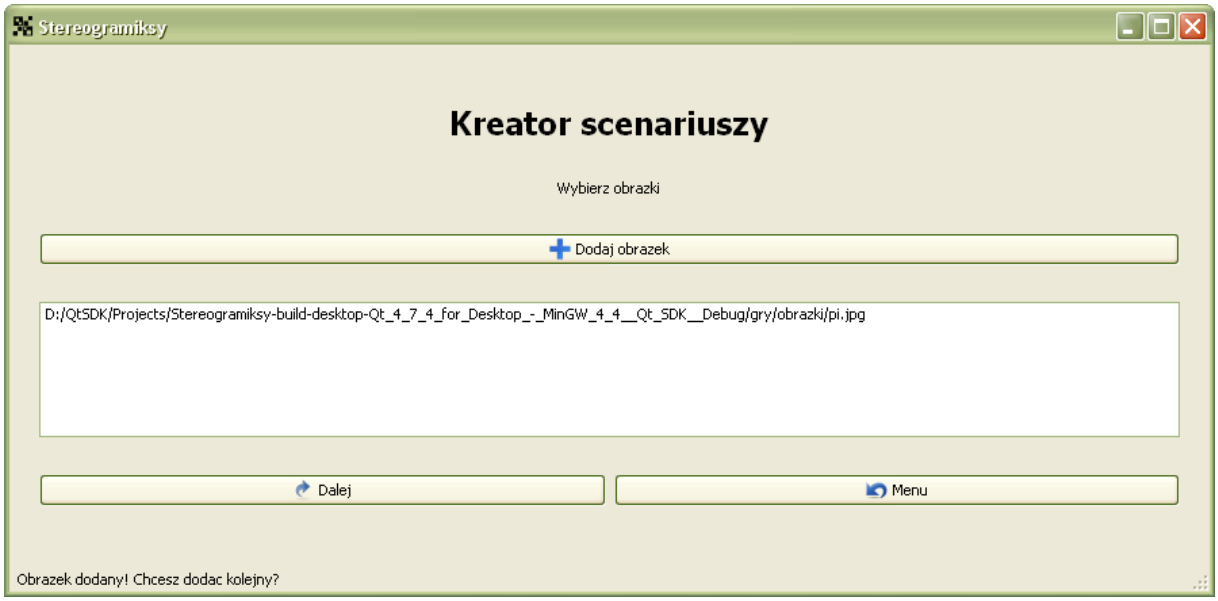
Okno przedstawiające końcowy rezultat gry. Jedyną opcją, jaką użytkownik ma do dyspozycji na tym poziomie, jest powrót do okna głównego.

PRZYCISK: Wróć do menu
Otwiera okno główne aplikacji.

LICZNIK: Twój wynik to
Prezentuje rezultat końcowy uzyskany przez gracza przez gracza, czyli sumę punktów zdobytych w każdym z etapów gry.

Okno kreatora scenariuszy I:





W pierwszym oknie kreatora scenariuszy użytkownik może sporządzić listę obrazów, które będą stanowiły kolejne etapy jego własnej gry. Wystarczy, że doda on nowy obrazek, a kiedy lista przyjmie satysfakcjonującą postać, przejdzie do kolejnego okna. Użytkownik ma również możliwość powrotu do głównego okna programu.

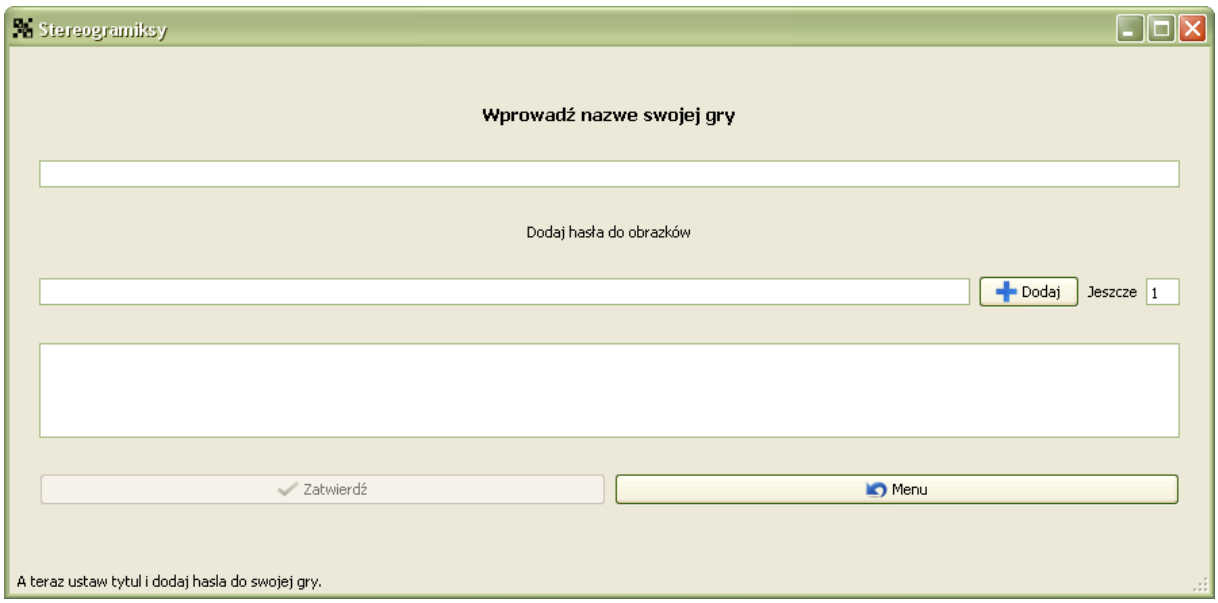
PRZYCISK: Dodaj obrazek
Dołącza do listy wybrany przez użytkownika plik graficzny w formacie „bmp”, „jpg” lub „png”.

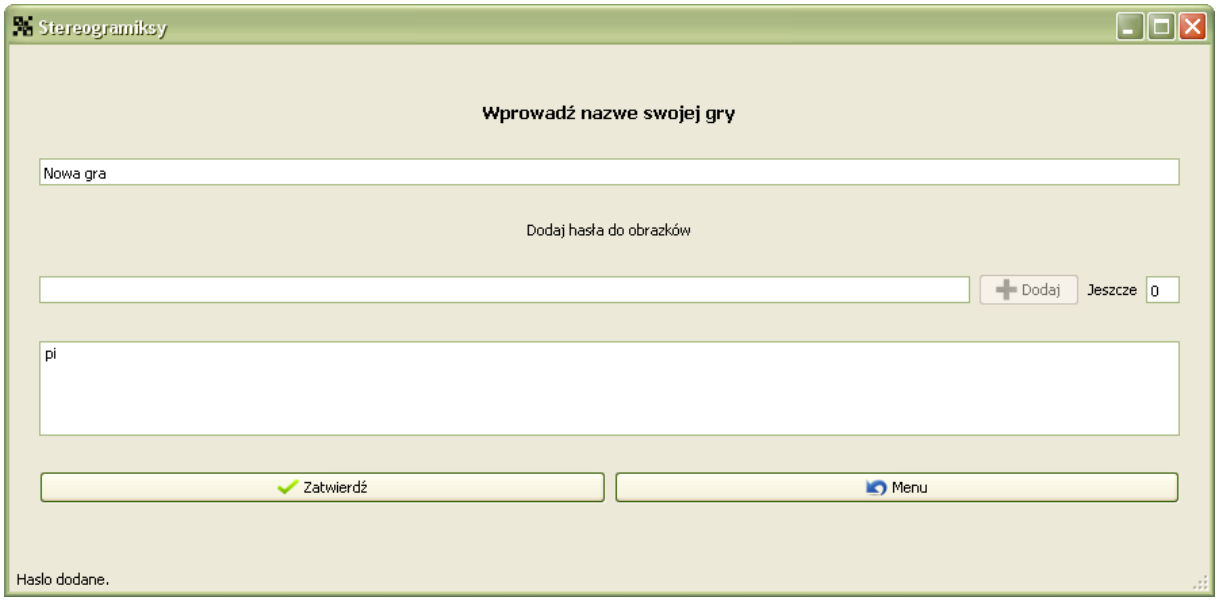
PRZYCISK: Menu
Otwiera główne okno aplikacji.

PRZYCISK: Dalej (dostępny po wczytaniu przynajmniej jednego pliku graficznego)
Zatwierdza sporządzoną listę i uruchamia okno kreatora scenariuszy II.

LISTA: po środku
Przedstawia aktualną listę obrazków stanowiących kolejne etapy tworzonej gry.

Okno kreatora scenariuszy II:





Drugie okno kreatora scenariuszy pozwala na powiązanie kolejnych obrazków na liście z hasłami pełniącymi rolę poprawnych odpowiedzi w następujących po sobie etapach gry. Użytkownik może, a nawet musi dodać tyle haseł, ile etapów dla danej gry przygotował. Dopiero potem może zatwierdzić wprowadzone hasła, tworząc tym samym własną grę. Ma również możliwość powrotu do głównego okna programu, nie kończąc tworzenia.

PRZYCISK: Dodaj (aktywny do czasu przypisania haseł wszystkim etapom)
Dołącza wprowadzone wyrażenie do listy haseł, wiąże go z odpowiadającym mu kolejnością obrazkiem na liście etapów z poprzedniego okna kreatora scenariuszy.

PRZYCISK: Menu
Otwiera okno główne aplikacji.

PRZYCISK: Zatwierdź (aktywny po przypisaniu haseł wszystkim etapom)
Zapisuje wprowadzone dane do odpowiedniego pliku tekstowego, z którego później będzie korzystał moduł gry przy jej wyborze i uruchamianiu. Otwiera okno główne aplikacji.

POLE EDYTOWALNE: Dodaj hasła do obrazków
Pole tekstowe, w którym użytkownik może wpisać dowolne wyrażenie, traktowane później jako prawidłowa odpowiedź w związanym z nim etapie gry.

POLE TEKSTOWE: Jeszcze
Informuje, ile etapów gry nie zostało opatrzonych w hasło z poprawną odpowiedzią.

LISTA: nad przyciskami
Zestawienie haseł dla odpowiadających im kolejnością obrazków – etapów gry.

Okno z informacjami o programie:

Okno dostarczające najbardziej podstawowych informacji odnośnie aplikacji, czyli jej wersję, główny cel oraz autorów. Dodatkowo zawiera też odnośnik do strony WWW projektu.
Nie stanowi reprezentacji graficznej żadnego modułu i nie daje możliwości zapoznania się ze stosowanymi w aplikacji procedurami. Możliwy jest jedynie powrót do okna głównego.

PRZYCISK: Menu
Otwiera okno główne aplikacji.

5. Moduły i zadania.

Całość pod względem funkcjonalnym można podzielić na 3 moduły:
- Generator stereogramów
- Gra
- Edytor scenariuszy

Generator stereogramów – to najważniejszy moduł aplikacji, który realizuje założenia opisane we wstępnych wymaganiach, w tym wszystkie podstawowe. Korzysta on z kluczowego w realizacji projektu algorytmu, w nieznaczny sposób warunkując jego działanie danymi wejściowymi, wprowadzonymi przez użytkownika. Wczytuje wybrany obrazek, na rzecz którego wywołuje algorytm odpowiedniego losowania punktów, a rezultat w postaci obrazu stereoskopowego umieszcza na panelu GUI. Ten sam moduł odpowiada za powtórne wywołanie procedur dla aktualizowanych wartości parametrów wejściowych, jak również za zapis do pliku graficznego w wybranym formacie i o podanych lub oryginalnych rozmiarach.

Gra – realizacja jednego z celów dodatkowych poszerzająca rozrywkową funkcję aplikacji. Polega ona na kolejnym wczytywaniu obrazków (w domyśle ze stereogramami) z edytowalnym polem tekstowym, w którym należy wpisać, co widać w „głębi" obrazu. Dane odnośnie plików graficznych, z jakich moduł gry korzysta, zapisane są w postaci list w plikach tekstowych podobnie, jak poprawne odpowiedzi. Za każde odgadnięte hasło gracz dostaje punkty, które są do siebie dodawane, a ich suma zostaje przedstawiona na koniec, kiedy nie ma już więcej obrazków. Nie ma ograniczeń w kwestii liczby dostępnych gier. Każda gra korzysta z plików graficznych, haseł oraz porządkującej je listy. Nie ma też żadnych limitów czasowych w trakcie trwania gry. Ważne jest, aby katalog z dostępnymi grami znajdował się w odpowiednim folderze programu.

Kreator scenariuszy – moduł realizujący kolejne dodatkowe założenie, czyli możliwość tworzenia własnej gry. Stanowi również uzupełnienie wyżej wymienionego modułu. Tworzenie nowej gry polega na wyborze kolejnych plików graficznych (w domyśle ze stereogramami) i dołączaniu do nich haseł z poprawną odpowiedzią. Jako wynik, moduł wygeneruje plik tekstowy, ustalający kolejność występowania po sobie obrazków oraz wiążący każdy obrazek z poprawną odpowiedzią, z którego później będzie korzystał moduł gry w trakcie wyboru i uruchamiania gry.

6. Środowisko, biblioteki, narzędzia.

Projekt w całości został napisany w języku C++. Wykorzystane środowisko to QtSDK, które pozwoliło na użycie biblioteki Qt do poprawnego wykonania wyświetlania obrazków oraz stworzenia graficznego interfejsu użytkownika. Użyta wersja Qt to 4.7.4. Kompilator, również zagwarantowany przez wcześniejszą instalację środowiska, to MinGW 4.4. Całość, czyli zarówno tworzenie GUI, jak i implementacja algorytmu, odbyły się z wykorzystaniem narzędzia QtCreator z własnym designerem GUI.

Za wykorzystaniem powyższych przemawiało przede wszystkim posiadane już doświadczenie w pisaniu kodu programów i projektowaniu części wizualnej.

Zainstalowane środowisko nie sprawiało żadnych problemów podczas realizowania projektu.

V. Podział pracy.

1. Główne zadania:

Dokładne określenie zakresu obowiązków jest trudne głównie przez wzgląd na fakt, że ten z czasem ulegał zmianom, przede wszystkim na etapie pisania kodu. Jak można zauważyć we wstępie dokumentu, każdy z członków zespołu ma mniejszy lub większy wkład w kod, GUI i każdy pełni też rolę testera. Stąd opis zadań i czasu na nie poświęconego będzie mocno przybliżony.

Michał Franczyk

Lider – odpowiedzialny za organizację projektu, sposobu komunikowania się pozostałych członków zespołu i najprawdopodobniej jedyna osoba w zespole, która widzi stereogramy

Zajęcie:	Czas:
Wyszukiwanie informacji przydatnych w realizacji projektu	3h
Translacja głównego algorytmu do postaci kodu	4h
Pisanie kodu funkcjonalnego	6h
Tworzenie GUI	4h
Wzbogacanie kodu algorytmu o nowe, dodatkowe funkcje	2h
Poprawki w GUI w kontekście nowych opcji	1h
Optymalizacja kodu	2h
Testowanie	1h

Łączny czas spędzony na realizowaniu zadań wynosi: 23 godzin.

Szymon Jagieła

Programista – realizator dodatkowych modułów

Zajęcie:	Czas:
Wyszukiwanie informacji przydatnych w realizacji projektu	3h
Pomoc w translacji głównego algorytmu do postaci kodu	1h
Tworzenie GUI	3h
Pisanie kodu dodatkowych modułów	4h 30min
Tworzenie GUI dodatkowych modułów	1h 30min
Rozplanowanie dokumentacji	30min
Testowanie	1h

Łączny czas spędzony na realizowaniu zadań wynosi: 14 godzin i 30 minut.

Michał Golonka

Pomocnik – dokumentowanie, parę pomysłów na rozszerzenie projektu i niewielkie zmiany w kodzie

Zajęcie:	Czas:
Wyszukiwanie informacji przydatnych w realizacji projektu	2h
Pomoc w translacji głównego algorytmu do postaci kodu	1h
Niewielkie wzbogacenie algorytmu wraz z ingerencją w kod	2h 30min
Poprawki GUI w związku z nową funkcją	30min
Testowanie	1h
Dokumentowanie	8h

Łączny czas spędzony na realizowaniu zadań wynosi: 15 godzin.

2. Ramy czasowe.

Do indywidualnych zadań należy jeszcze dodać czas poświęcony na organizację zespołu, omówienie założeń, celów, wybór środowiska, a także sposobu komunikowania się w trakcie realizacji projektu. Dopiero suma tych wszystkich czynników daje przybliżony czas, jaki był potrzebny na wykonanie całości.

Zajęcie:	Czas:
Organizacja zespołu i sposobu komunikowania się	2h
Omówienie założeń i wybór środowiska	2h 30min
Wyszukiwanie przydatnych informacji	8h
Translacja głównego algorytmu do postaci kodu	6h
Pisanie całego kodu funkcjonalnego	19h
Tworzenie GUI wszystkich modułów	10h
Testowanie	3h
Dokumentowanie	8h 30min

Łączny czas poświęcony na realizację projektu wynosi: 57 godzin.

VI. Algorytm.

1. Zasada działania.

Stosowany algorytm bada piksel obrazka w skali szarości pod kątem jego jasności. Na podstawie tej informacji dopasowuje do niego głębokość. Wartość 1 oznacza piksel czarny, zaś 0 to piksel biały. Oznacza to, że ciemniejsze fragmenty będą znajdowały się „bliżej" obserwującego.

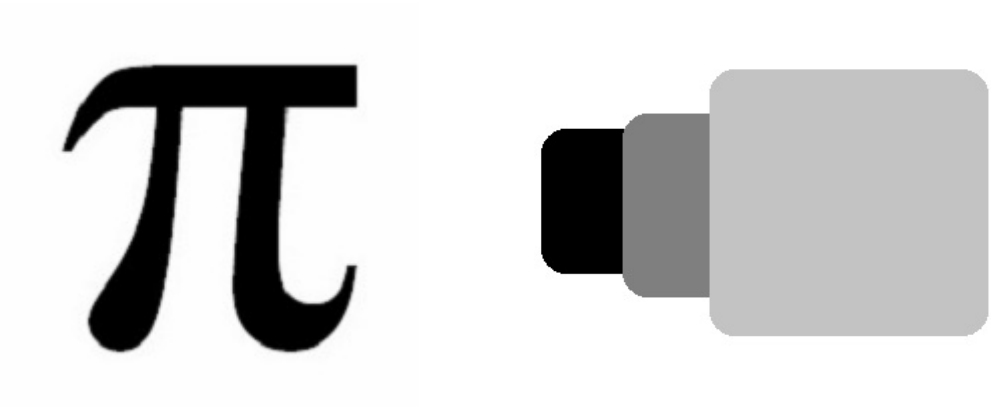
W dalszej kolejności piksel zostaje rozdwojony na piksel, który będzie oglądany lewym okiem oraz piksel, na który obserwator będzie patrzył prawym. Kluczowa jest tutaj odległość między nimi, która zależy od znanej już głębokości podwojonego piksela. Należy zwrócić uwagę, że przy takiej operacji część pikseli pokryje inne. Jest to całkiem logiczne, biorąc pod uwagę fakt, że jako końcowy efekt mamy otrzymać obrazek dający złudzenie trójwymiarowości, a więc taki, w którym coś jest z przodu, a coś jest z tyłu. To, co jest z przodu pokryje część powierzchni znajdującej się dalej, czyniąc ją niewidoczną. Algorytm radzi sobie z tym w taki sposób, że nie generuje tej „zasłoniętej" powierzchni, a zamiast tego usuwa piksele, które miałyby ją utworzyć.

W końcowej fazie losowany jest kolor piksela, czarny lub biały. Operacja dotyczy tylko odpowiednich pikseli, tak, aby nie zepsuć pożądanego efektu 3D. Dlatego też na koniec otrzymujemy obrazek, który na pierwszy rzut oka stanowi chaotyczne rozłożenie czarnych punktów na białym tle, ale przy odpowiednim rzucie „obojgiem oczu", obserwator może dostrzec głębię oraz elementy znajdujące się bardziej z przodu i bardziej z tyłu.

Algorytm został zaczerpnięty z:
„Displaying 3D Images: Algorithms for Single Image Random Dot Stereograms"
Autorzy: Harold W. Thimbleby, Stuart Inglis, Ian H. Witten.

Dodatkowo wyposażono go w możliwość generowania obrazków w innym kolorze niż biały, jak również opcję dostosowania do DPI komputera i rozstawu oczu obserwującego.

Przykładowe obrazki na wejście:



Kod:

```
/// wywołanie funkcji obliczającej "głębokość" pikseli na skali szarości
///( czarny 1, biały 0 ), im ciemniejszy tym bliżej nas
    calculateImageDepth(imageDepth,convex);

    /// wysłanie do status bara, w głownym oknie aplikacji, wiadomości
    emit setStatusBarLabel("Generowanie");

    /// początek algorytmu generującego stereogramy
    /// przechodzimy po wszystkich pikselach, po pikselach poziomych
    /// tak aby zastosować algorytm do wszystkich linii oddzielnie
    /// x, y współrzędne danego punktu ( piksela )
    for(int y=0;y<_heightOfImage_Y;++y)
    {
        /// tablica kolorów danych pixeli
        unsigned int colorOfPixel[ widthOfImage X];

/// tablica pixeli, które wskazują na te pixele
/// które mają mieć wymuszony dany kolor ( czarny lub biały )
/// [wszystko w zależności od tego, czy mamy informację na temat "głębokości" danego piksela]
        int samePixels[ heightOfImage Y];

 /// punkt - przechowuje informacje na temat rozdzielenia
 /// dwóch identycznych pikseli odpowiadających rozdzieleniu "stereo" tych punktów
 /// @note rozdzielenie "stereo" efekt wizualny - wykorzystywany do generowania stereogramów
        int stereoSeparationOfPoint;

        /// "lewy" i "prawy" piksel
        /// muszą być tego samego koloru
        /// - odpowiadają pikselowi widocznemu przez lewe i analogicznie prawe oko
        int leftEye, rightEye;

        /// inicjalizacja tablicy pikseli samePixels
        /// początkowo przypisujemy pikselom ich własne wartości
        for(int x=0;x<_widthOfImage_X;++x)
            samePixels[x]=x;

        for(int x=0;x< widthOfImage X;++x)
        {
    /// przypisanie - rozdzielenie dwóch punktów
            stereoSeparationOfPoint = separateSomething(imageDepth[x][y]);
    /// przypisanie wartości do zmiennych przechowujących
    /// informacje o pikselu lewym i prawym
    /// - piksel lewy i prawy muszą być takie same (albo przynajmniej powinny takie być ...)
            leftEye = x - stereoSeparationOfPoint/2;
            rightEye = leftEye + stereoSeparationOfPoint;

            /// dalsza część algorytmu
            /// - usuwanie niewidocznych powierzchni
            if(0 <= leftEye && rightEye <  widthOfImage X)
            {
        /// zmienna odpowiedzialna za usuwanie niewidocznych ( ukrytych ) powierzchni
               int isVisible;

        /// zmienna tymczasowa, porządkująca kod i algorytm
        /// [będziemy sprawdzać piksele x+tmp i x-tmp - czy nie są zakryte, bądź takie same]
                int tmp=1;

                /// zmienna przechowująca wartość odległości od "ściany" stereogramu
                /// zależnie od wartości imageDepth - czyli głębi na skali szarości
                float rayOfImageDepthAtPoint;

         /// pętla odpowiadająca za przejście i usunięcie wszystkich ukrytych powierzchni
                do
                {
                    /// implementacja wzoru wyliczającego odległości ( promień )
                    rayOfImageDepthAtPoint = imageDepth[x][y] + 2*(2 -
_depthOfField*imageDepth[x][y])*tmp/(_depthOfField*_eyeSeparation);

                    /// jeśli dany piksel jest zasłonięty to isVisible = 0
                    isVisible = imageDepth[x-tmp][y] < rayOfImageDepthAtPoint &&
imageDepth[x+tmp][y] < rayOfImageDepthAtPoint;
```

```
                /// post-inkrementacja tmp
                /// - będziemy sprawdzać dla kolejnych sąsiednich pikseli czy się pokrywają
                   tmp++;
               }
               while (isVisible && rayOfImageDepthAtPoint < 1);

               /// sprawdzamy czy piksele są takie same - jeśli są widoczne
               if (isVisible)
               {
                   int left_N_right = samePixels[leftEye];
                   while (left_N_right != leftEye && left_N_right != rightEye)
                   {
                       /// następnie porównujemy piksele
                       /// do momentu aż samePixels[leftEye] = leftEye
                       /// lub samePixels[leftEye] = rightEye
                       /// jeśli nie są - zamieniamy je miejscami
                       if (left_N_right > rightEye)
                       {
                           leftEye = left_N_right;
                           left_N_right = samePixels[leftEye];
                       }
                       else
                       {
                           samePixels[leftEye] = rightEye;
                           leftEye = rightEye;
                           left_N_right = samePixels[leftEye];
                           rightEye = left_N_right;
                       }
                   }

               /// odnotowujemy fakt, że piksele są takie same
               samePixels[leftEye] = rightEye;
               }
           }
       }

       /// końcowa część algorytmu
       /// ostatnia pętla po której odpowiednim pikselom
       /// zostaje nadany kolor
       for (int x=_widthOfImage_X-1 ; x>= 0 ; x--)
       {
           /// losowo nadajemy im wartości
           /// zamiarem algorytmu jest uzyskanie koloru czarnego lub białego
           /// dokładniej dwóch różnych kontrastowych kolorów
           /// - czarny powinien zostać
           if (samePixels[x] == x) colorOfPixel[x] = qrand()&1;
           else colorOfPixel[x] = colorOfPixel[samePixels[x]];

/// switch odpowiedzialny za nadanie innego koloru stereogramowi ( niż biały )
           /// - poprzez wybranie odpowiedniej wartosci color przez uzytkowika
           switch (color)
           {
case 0: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*16775930);/* White */break;
case 1:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*14210000);/* Gray */break;
case 2:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*11500000);/* Purple */break;
case 3:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*11800000);/* Violet */break;
case 4: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*14245000);/* Pink */break;
case 5: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*14300000);/* Scarlet */break;
case 6: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*16000000);/* Red */break;
case 7:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*14250000);/* Orange */break;
case 8:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*9000000);/* Brown */break;
case 9: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*14272000);/* Yellow */break;
case 10: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*9300000);/* Pistac. */break;
case 11: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*125000);/* Green */break;
case 12:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*1830000);/* Sea gr. */break;
case 13:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*1900000);/* Sky bl. */break;
case 14: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*300000);/* Indigo */break;
case 15: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*255);/* Blue */break;
case 16: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*16000000/_imageCopy-
>width()*x); /* Multi-color columns */ break;
case 17:  imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*16000000/ imageCopy-
>height()*y); /* Multi-color rows */   break;
default: _imageGeneratedStereogram->setPixel(x,y,colorOfPixel[x]*16775930);/* White */break;
           }
   }
```

VI. Kodowanie.

1. Dokumentacja kodu.

Szczegółowe informacje na temat struktury kodu, takie jak opis plików nagłówkowych, plików źródłowych, klas, funkcji i zmiennych można znaleźć w oddzielnie załączonych dokumentach „html” wygenerowanych programem Doxygen.

Pliki nagłówkowe:
- mainwindow.h
- stereogramgenerator.h

Pliki źródłowe:
- main.cpp
- mainwindow.cpp
- stereogramgenerator.cpp

2. Kody źródłowe z opisami.

stereogramgenerator.h

```
#include <QObject>
#include <QImage>
#include <QDebug>
#include <QPainter>

/**
 * StereogramGenerator - klasa odpowiadająca za generowanie stereogramów
 * Zawiera algorytm oraz funckje pomocnicze w tworzeniu stereogramu z dostarczonego obrazka
 * Opis algorytmu znajduje się dokumentacji.
 * @file stereogramgenerator.h
 * @note Klasa generująca stereogramy
 * @see MainWindow
 */
class StereogramGenerator : public QObject
{
    Q_OBJECT
public:
        /**
         * StereogramGenerator - wstępna inicjalizacja argumentów, konstruktor
         * Konstruktor klasy MainWindow
         * @see setDefault();
         * @see changeDefault();
         */
        explicit StereogramGenerator(QObject *parent = 0);

        /**
         * Destruktor klasy StereogramGenerator
         */
        ~StereogramGenerator();

        /**
         * setImage - ustawia plik, z którego będzie tworzony stereogram
         * @param *img - przekazany obrazek do wygenerowania ( *img = *imageCpy )
         * @see getImage();
         * @see generate();
         * @return void
         */
        inline void setImage(QImage *img){ imageToGenerate = img;}
```

```
        /**
         * getImage - zwraca wygenerowany plik ( stereogram )
         * @see setImage();
         * @see generate();
         * @return QImage - wygenerowany stereogram
         */
        inline QImage* getImage(){return _imageGeneratedStereogram;}

        /**
         * generate - główna funkcja klasy StereogramGenerator, zawiera w sobie algorytm
generujący stereogramy oraz wywołuje funckje pozwalające na zmianę rozmiaru, koloru,
wklęsłości, rysowania pomocniczych kropek
         * @param convex - zmienna odpowiadająca za "wklęsłość" - to czy wygenerowany obrazek
będzie widoczny jako zwykły obiekt lub jako wycięty.
         * @param color - odpowiada za kolory w jakich ma zostać wygenerowany stereogram
         * @param circles -  odpowiada za to czy na obrazku mają się pojawić pomocnicze w
zobaczeniu stereogramu kółka
         * @param size - odpowiada za rozmiar w jakim ma być wygenerowany stereogram -
głownie na potrzeby zapisu do pliku
         * @param keepAspectRatio - odpowiada za to czy obrazek po zmianie rozmiaru ma się
dopasować do przyjętych proporcji czy porzucić swoje proporcje i dopasować się rozmiaru (
rozciągnąć, spłaszczyć )
         * @see setImage();
         * @see drawCirclesOnImage();
         * @see calculateImageDepth();
         * @see roundSomething();
         * @see separateSomething();
         * @see setStatusBarLabel();
         * @return void
         */
        void generate(int convex, int color=false, bool circles=false, int size=4, int
keepAspectRatio=true);

        /**
         * setDefault - ustawia wartości domyślne klasy StereogramGenerator
         * @see generate();
         * @see changeDefault();
         * @return void
         */
        void setDefault(int DPI, double distanceBetweenEyes);

        /**
         * changeDefault - zmienia wartości domyślne
         * @param currentDPI - ustawia DPI wybrane przez użytkownika
         * @param currentDistanceBetweenEyes - ustawia odległość między oczami ( w calach )
wybraną przez użytkownika.
         * @see generate();
         * @see changeDefault();
         * @return void
         */
        void changeDefault(int currentDPI, int currentDistanceBetweenEyes);

    signals:

        /**
         * setStatusBarLabel - sygnał wysyłany do klasy MainWindow, w która obsługuje zmienną
ui - odpowiedzialną za GUI
         * @param tmp - string, ciąg znaków jaki ma się pojawić na status barze
         * @return void
         */
        void setStatusBarLabel(QString tmp);

    public slots:
        // empty
```

```
    private:

        /**
         * drawCirclesOnImage - funkcja rysująca na obrazku pomocnicze kropki
         * ( zmienne pomagają określić w jakim miejscu mają się znaleźć kropki )
         * @param maxX - szerokość obrazka
         * @param maxY - wysokość obrazka
         * @see generate();
         * @see calculateImageDepth();
         * @return void
         */
        void drawCirclesOnImage(int maxX, int maxY);

        /**
         * calculateImageDepth - funkcja obliczająca "głębokość" ( w skali szarości ) piksel
po pikselu
         * ( dla koloru białego - 0, dla koloru czarnego - 1 )
         * @param **imageDepth - tablica, która zostanie uzupełniona wartościami danych
pikseli na skali szarości
         * @param convex - wklęsłość
         * @see generate();
         * @see drawCirclesOnImage();
         * @return void
         */
        void calculateImageDepth(double **imageDepth, int convex);

        /**
         * roundSomething - funkcja zaokrąglająca daną liczbę ( np. typu double ) do int'a
         * @param something - zmienna do zaokrąglenia
         * @see generate();
         * @see drawCirclesOnImage();
         * @return int - zwraca zaokragloną liczbę
         */
        int roundSomething(double something);

        /**
         * separateSomething - funkcja ( rozdzielająca ) pozycję, pobiera wartość piksela w
skali szarości
         * a następnie zwraca odległość w jakiej powinien znaleźć od "ściany" obrazka
         * @param something - pozycja
         * @see generate();
         * @see roundSomething();
         * @return int - zwraca rozdzieloną liczbę
         */
        int separateSomething(double something);

        /**
          * Zmienna  imageToGenerate - przechowuje obrazek, z którego będzie generowany
stereogram
          * Zmienna prywatna.
          */
        QImage *_imageToGenerate;

        /**
          * Zmienna  imageCopy - przechowuje kopię oryginalnego obrazka ( na potrzeby zmiany
rozmiaru itd. )
          * Zmienna prywatna.
          */
        QImage *_imageCopy;

        /**
          * Zmienna _imageGeneratedStereogram - przechowuje wygenerowany stereogram
          * Zmienna prywatna.
          */
        QImage * imageGeneratedStereogram;

        /**
          * Zmienna _widthOfImage_X - przechowuje szerokość ( w pikselach ) przeskalowanego
obrazka ( generowanego stereogramu )
          * Zmienna prywatna.
          */
        int  widthOfImage X;
```

```
        /**
          * Zmienna _widthOfImage_X - przechowuje wysokość ( w pikselach )  przeskalowanego
obrazka ( generowanego stereogramu )
          * Zmienna prywatna.
          */
        int  heightOfImage Y;

        /**
          * Zmienna _wygenerowano - true jeśli wczytano obrazek - pomaga w zarządzaniu
pamięcią
          * Zmienna prywatna.
          */
        bool _wygenerowano;

        /**
          * Zmienna  distanceBetweenEyes - odległość między oczami ( w calach )
          * Zmienna prywatna.
          */
        double _distanceBetweenEyes;

        /**
          * Zmienna  DPI - DPI ekranu, zmienna może być zmieniona przez użytkownika,
domyślnie 96
          * Zmienna prywatna.
          */
        double  DPI;

        /**
          * Zmienna _depthOfField - głębia widzialnego pola, ułamek ( 1/3 ) z widzialnego
dystansu
          * Zmienna prywatna.
          */
        double  depthOfField;

        /**
          * Zmienna  eyeSeparation - zmienna odpowiadająca rozdzieleniu "oczu", pola
widzialnego
          * zależna od  distanceBetweenEyes oraz  DPI
          *  eyeSeparation = roundSomething( distanceBetweenEyes* DPI)
          * Zmienna prywatna.
          */
        int  eyeSeparation;

        /**
          * Zmienna  _farOfDots - wykorzystywana przy rysowaniu pomocniczych kropek -
wyznacza ich odległość
          * Zmienna prywatna.
          */
        int  farOfDots;
};
```

mainwindow.h

```
#include <QMainWindow>
#include <QFileDialog>
#include <QStringListModel>
#include <QImageReader>
#include "stereogramgenerator.h"

/**
 * Ui - namespace - do używania obiektów stworzonych za pomocą Designer'a w QtCreatorze
 * @see mainwindow.ui
 * @see MainWindow
 */
namespace Ui
{
    class MainWindow;
}
```

```
/**
 * MainWindow - Główna Klasa .
 * Klasa, kontrolująca wszelkie działania aplikacji.
 * @note MainWindow musi działać.
 * @file mainwindow.h
 */
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:

    /**
     * MainWindow - ręczne dodawanie/ustawianie elementów.
     * Konstruktor klasy MainWindow
     */
    explicit MainWindow(QWidget *parent = 0);

    /**
     * Destruktor klasy MainWindow
     */
    ~MainWindow();

private slots:

    /** @brief funkcja odpowiadająca za wyświetlenie na labelu informacji o wczytanym obrazku
      * @return void
     */
    void setLabel_info(int w, int h, float size, bool allgray);

    /** @brief funkcja odpowiadająca za wyświetlenie na status barze aktualnych
informacji/podpowiedzi
      * @return void
     */
    void onStatusBarChanged(QString tmp);

    /** @brief funkcja obsługująca przycisk przenoszący użytkownika do odpowiedniej
generowania stereogramu
      * @return void
     */
    void on_pushButton_Generate_clicked();

    /** @brief funkcja obsługująca przycisk otwierający OpenFileDialog
      * @return void
     */
    void on_pushButton_Open_clicked();

    /** @brief funkcja obsługująca przycisk zamykający program
      * @return void
     */
    void on_pushButton_Koniec_clicked();

    /** @brief funkcja obsługująca przycisk powracający do głównego menu z zakładki
generowanie stereogramu
      * @return void
     */
    void on_pushButton_clicked();

    /** @brief Wybór zakładki 'Gra' z menu głównego programu.
      * @return void
     */
    void on_pushButton_Games_clicked();

    /** @brief funkcja obsługująca przycisk otwierający SaveFileDialog - zapisuje obrazek
      * @return void
     */
    void on_pushButton_Save_clicked();

    /** @brief W zakładce wyboru gier wybieramy grę, w którą chcemy zagrać.
      * @return void
     */
    void on pushButton wg Graj clicked();
```

```
    /** @brief Powrót do menu głównego z zakładki wybory gier.
     */
    void on_pushButton_wg_Menu_clicked();

    /** @brief Akcja na kliknięcie odpowiedzi do gry.
      * @return void
     */
    void on_pushButton_game_Ok_clicked();

    /** @brief funkcja obsługująca przycisk powracający do głównego menu z zakładki z
wynikami gry
      * @return void
     */
    void on_pushButton_2_end_menu_clicked();

    /** @brief funkcja obsługująca check odpowiedzialny za wygenerowanie stereogramu z
pomocniczymi kropkami
      * @return void
     */
    void on_checkBox_clicked();

    /** @brief Wybór zakładki 'Stwórz scenariusz' w menu głównym.
      * @return void
     */
    void on pushButton Scenariusz clicked();

    /** @brief Powrót do menu z zakładki scenariusz.
      * @return void
     */
    void on_pushButton_scen_menu_clicked();

    /** @brief Przycisk akcji dodający obrazek do scenariusza.
      * @return void
     */
    void on_pushButton_scen_add_clicked();

    /** @brief Dodanie wszystkich obrazków i przejście do kolejnego etapu.
      * @return void
     */
    void on_pushButton_scen_dalej_clicked();

    /** @brief Powrót do menu z zakładki stworzenia scenariusza.
      * @return void
     */
    void on_pushButton_scen2_menu_clicked();

    /** @brief Dodanie hasła do gry przy tworzeniu scenariusza.
      * @return void
     */
    void on pushButton scen2 add clicked();

    /** @brief Zakończenie tworzenia scenariusza i zapis do pliku.
      * @return void
     */
    void on pushButton scen2 ok clicked();

    /** @brief funkcja obsługująca combobox odpowiedzialny za zmianę kolorów na stereogramie
      * @return void
     */
    void on comboBox activated(int index);

    /** @brief Wybór zakładki 'o programie' z menu głównego.
      * @return void
     */
    void on pushButton Oprogramie clicked();

    /** @brief Powrót do menu z zakładki o programie.
      * @return void
     */
    void on_pushButton_oprogramie_menu_clicked();
```

```
    /** @brief funkcja obsługująca combobox odpowiedzialny za zmianę DPI generowanego
stereogramu
      * @return void
     */
    void on comboBox DPI activated(int index);

    /** @brief funkcja obsługująca combobox odpowiedzialny za rozstawu oczu generowanego
stereogramu
      * @return void
     */
    void on_comboBox_RozstawOczu_activated(int index);

    /** @brief funkcja obsługująca checkbox odpowiedzialny za zmianę "wklęsłości" stereogramu
- więcej w dokumentacji.
      * @return void
     */
    void on_checkBox_wkleslosc_clicked();

    /** @brief funkcja obsługująca checkbox odpowiedzialny za to czy obrazek ma się
dopasowywać do okna czy też pozostać w oryginalnym kształcie
      * @return void
     */
    void on_checkBox_dopasujDoOkna_clicked();

private:

    /**
     * update - odświeżająca okienko - dopasowuje rozmiar wczytanego, a następnie
wygenerowanego pliku ( stereogramu )
     * @param dopasuj - czy obrazek ma być dopasowany do okna ( domyślna wartość - tak )
     * @see openFile();
     * @see saveFile();
     * @see resizeEvent();
     * @see wczytajPlik();
     * @return void
     */
    void update(bool dopasuj=true);

    /**
     * openFile - okienko dialogowe do wyboru pliku.
     * @see wczytajPlik();
     * @see update();
     * @see saveFile();
     * @see resizeEvent();
     * @return void
     */
    void openFile();

    /**
     * saveFile - zapisuje plik ( stereogram ) - w określonej rozdzielczości, wybranym
rozszerzeniem oraz innymi ustawieniami
     * @see update();
     * @see saveFile();
     * @see resizeEvent();
     * @see wczytajPlik();
     * @return void
     */
    void saveFile();

    /**
     * wczytajPlik - metoda wczytująca obrazek
     * @param fileName - string nazwy pliku.
     * @see update();
     * @see saveFile();
     * @see resizeEvent();
     * @see wczytajPlik();
     * @return bool
     */
    bool wczytajPlik(QString fileName);
```

```
    /**
     * resizeEvent - event odpowiadający za updatowanie rozmiaru obrazka
     * @param event - rozszerzenie obrazka
     * @see update();
     * @see saveFile();
     * @see resizeEvent();
     * @see wczytajPlik();
     * @return void
     */
    void resizeEvent(QResizeEvent *);

    /**
     * setElementsDisabled - pozwala na łatwe "włączanie" lub "wyłączanie" elementów
     * @param disabled - czy dane elementy mają być włączone bądź nie
     * @see update();
     * @see saveFile();
     * @see resizeEvent();
     * @see wczytajPlik();
     * @return void
     */
    void setElementsDisabled(bool disabled);

    /**
      * Zmienna *ui - pozwala się odwoływać do obiektów stworzonych przez Designera
      * Zmienna prywatna.
      */
    Ui::MainWindow *ui;

    /**
      * Zmienna *fileName - wczytuje ścieżkę dostępu do pliku oraz jego nazwę.
      * Zmienna prywatna.
      */
    QString *fileName;

    /**
      * Zmienna *imageOrg - wczytany oryginalny obrazek
      * Zmienna prywatna.
      */
    QImage *imageOrg;

    /**
      * Zmienna *imageCpy - na początku przechowuje kopię imageOrg a potem sam stereogram.
      * Zmienna prywatna.
      */
    QImage *imageCpy;

    /**
      * Zmienna *imageToSave - przechowuje obrazek gotowy do zapisania.
      * Zmienna prywatna.
      */
    QImage *imageToSave;

    /**
      * Zmienna scaledImage - przechowuje kopię imageCpy - odpowiednio dopasowaną do rozmiaru
okna
      * Zmienna prywatna.
      */
    QImage scaledImage;

    /**
      * Zmienna wczytano - na wejściu false, po wygenerowaniu true | pomaga w optymalizacji
zużycia pamięci
      * Zmienna prywatna.
      */
    bool wczytano;

    /**
      * Zmienna _stereogramGenerator - obiekt klasy StereogramGenerator - odpowiedzialny za
obsługę generowania stereogramu
      * Zmienna prywatna.
      */
    StereogramGenerator  stereogramGenerator;
```

```
    /**
      * Zmienna tytulGra - przechowuje nazwę gry
      * Zmienna prywatna.
      */
    QString tytulGra;

    /**
      * Zmienna ileObrazkowGra - przechowuje liczbę obrazków w grze
      * Zmienna prywatna.
      */
    int ileObrazkowGra;

    /**
      * Zmienna adresObrazkaGra - przechowuje adres do obrazka w grze
      * Zmienna prywatna.
      */
    QStringList adresObrazkaGra;

    /**
      * Zmienna hasloObrazkaGra - przechowuje odpowiedz do obrazka w grze
      * Zmienna prywatna.
      */
    QStringList hasloObrazkaGra;

    /**
      * Zmienna punktyGra - przechowuje ilość zgromadzonych punktów w grze
      * Zmienna prywatna.
      */
    int  punktyGra;

    /**
      * Zmienna ktoryObrazekGra - przechowuje aktualny obrazek do wyswietlenia w grze
      * Zmienna prywatna.
      */
    int ktoryObrazekGra;

    /**
      * Zmienna fileListScene - przechowuje adresy obrazków w generatorze scenariuszy
      * Zmienna prywatna.
      */
    QStringList fileListScen;

    /**
      * Zmienna haslaListScene - przechowuje odpowiedzi do obrazków w generatorze scenariuszy
      * Zmienna prywatna.
      */
    QStringList haslaListScen;

    /** @brief Wczytanie parametrów gry.
     * @param gra Nazwa gry, w którą chcemy zagrać.
     * @see statsGame();
     * @see showGame();
     * @see endGame();
     * @return void
     */
    void loadGame(QString);

    /** @brief Uaktualnienie statystyk  oraz punktów gry.
     *  @param wygrana Sprawdza czy odpowiedź jest prawidłowa.
     * @return void
     */
    void statsGame(int);

    /** @brief Wyświetlenie obrazka w grze.
      * @see endGame();
      * @return void
     */
    void showGame();


    /** @brief Wyswietla zakladke podsumowania gry.
      * @return void
     */
    void endGame();
};
```

VII. Testowanie.

1. Testy modułów.

Sprawdzanie poprawności działania dla kolejnych modułów przebiegały w różny sposób, za pomocą innych procedur.

Generator stereogramów – testowanie głównego komponentu oparte było na sprawdzaniu poprawności wszystkich dostępnych w nim funkcji w różnej kolejności, czyli np. wczytywanie obrazka „jpg", potem zamiana koloru, zmiana DPI i rozstawu oczu, wyrysowywanie kropek i dostosowywanie do rozmiaru okna, rozciąganie okna GUI dla zaobserwowania efektów działania. Potem wczytanie nowego obrazka, ale w innym formacie (np. „bmp") bez wracania do menu. Powtórzenie zmian parametrów, jednak w innej kolejności. Równocześnie testom poddawany był zapis obrazka, raz z oryginalnymi rozmiarami, raz z wybranymi w opcjach. Sprawdzane były obrazy o formatach „bmp", „jpg" oraz „png" o różnych rozmiarach oryginalnych. Na podstawie wyników stwierdzono, że moduł działa zgodnie z założeniami.

Gra – moduł gry sprawdzano na przykładzie kilku przygotowanych gier, każda o różnej długości, oparta na różnych plikach graficznych i hasłach. Ostatecznie nie stwierdzono odstępstw od postawionych wymagań.

Kreator scenariuszy – sprawdzony przez przygotowanie własnej gry, ładowanie obrazków w różnych formatach („bmp", „jpg", „png") o innych rozmiarach każdy, a następnie dodawanie haseł z prawidłowymi odpowiedziami i ostatni krok w sprawdzaniu, czy moduł działa prawidłowo, czyli uruchomienie modułu gry na utworzonym scenariuszu. Ostatnie wyniki okazały się dostatecznie zadowalające, aby stwierdzić poprawność działania.

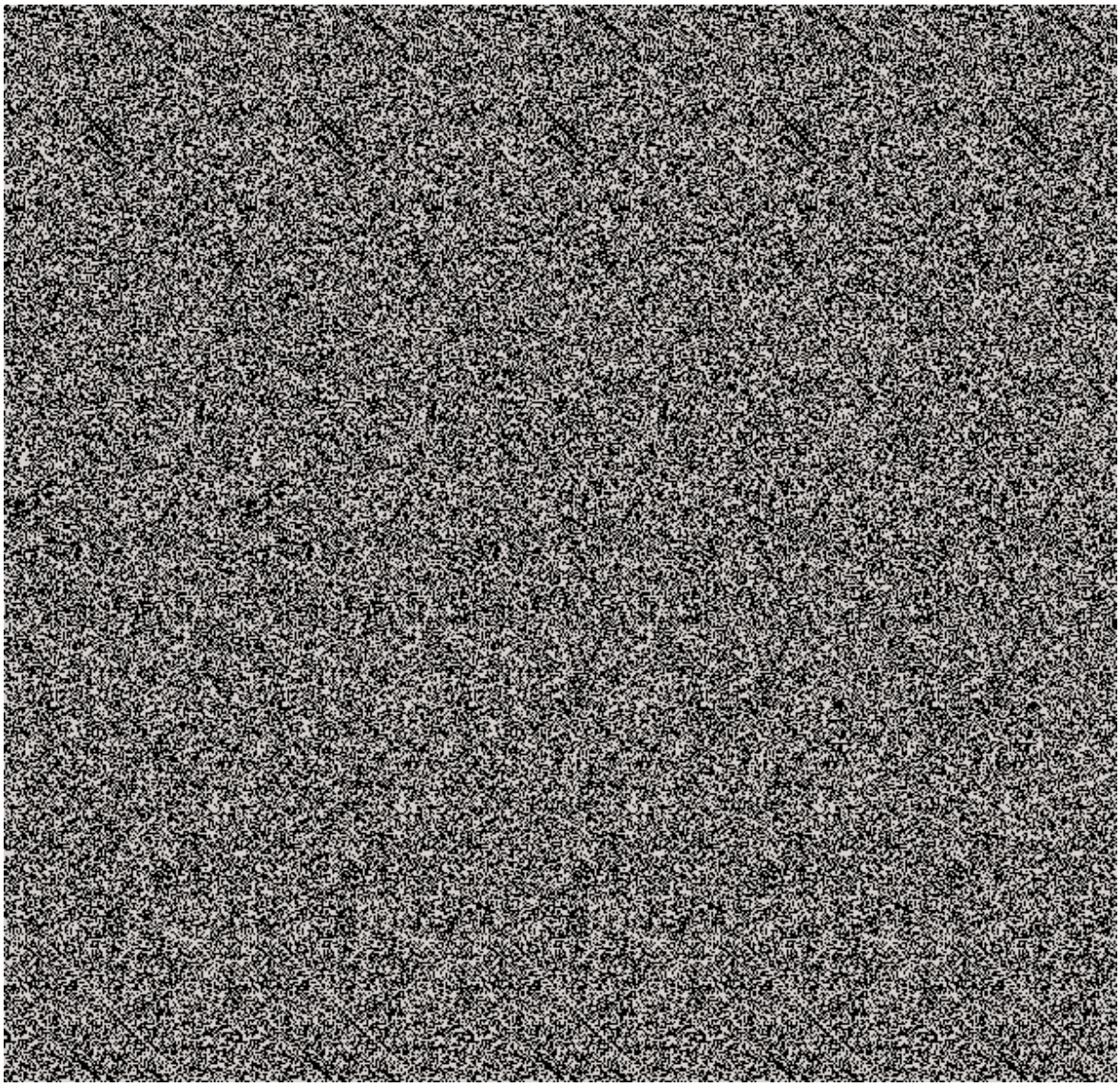
2. Testy ogólne.

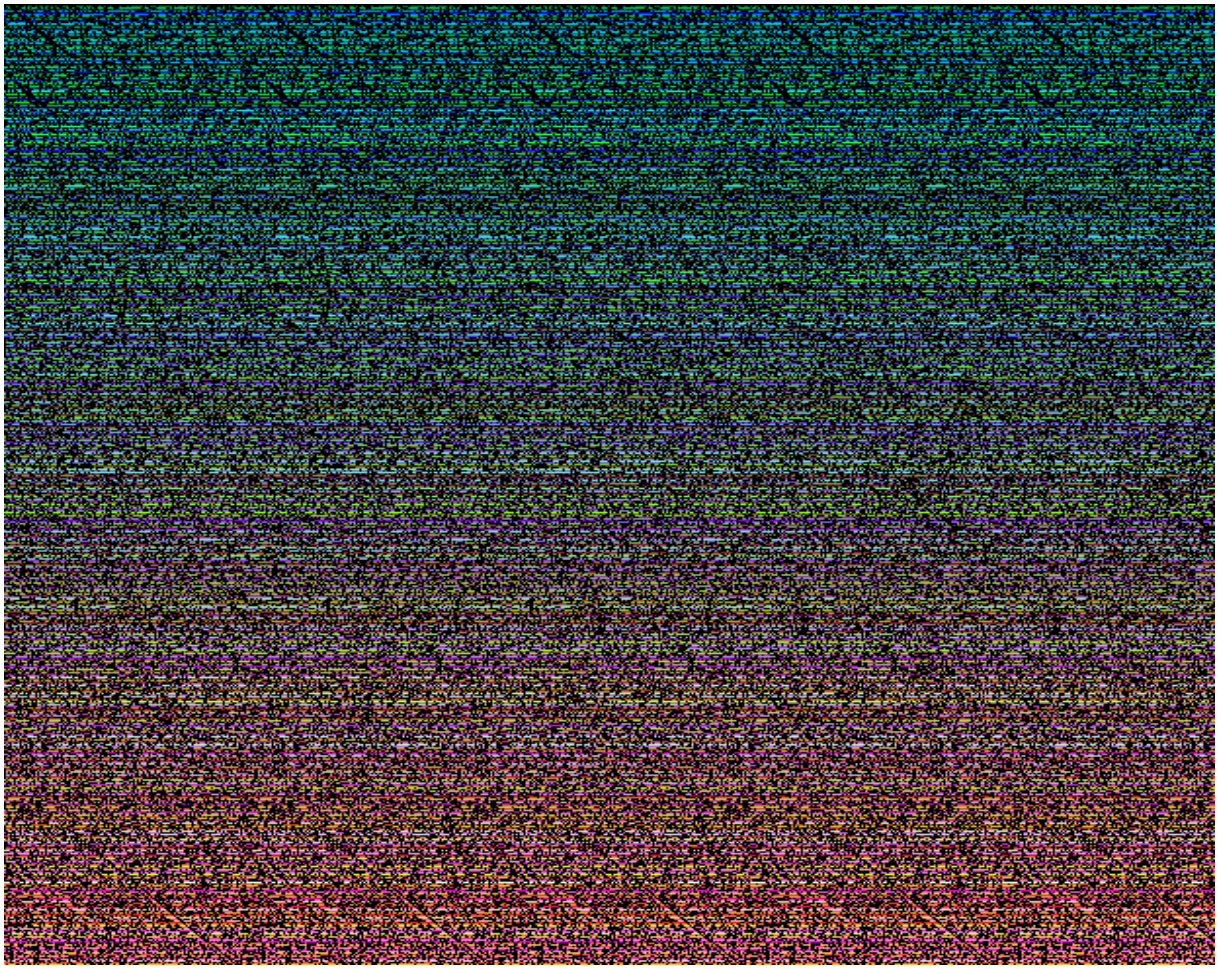
W początkowych fazach weryfikacji przynosiła ona wiele niepożądanych wyników, zaczynając od tych najbardziej elementarnych, wynikających z niewłaściwego kodowania, aż po te, których pozbycie się wymagało powtórnego przemyślenia pisanych procedur. Wśród nich można wymienić problem z modułem gry, który niezależnie od wyboru scenariusza sięgał po ten sam obrazek i tę samą odpowiedź. Dużo czasu zajęła kwestia przemyślenia skalowania wczytywanego przez generator obrazka, gdyż dla pewnych rozmiarów dawał wyniki niezadowalające. Wśród błędów związanych z GUI pojawiły się takie jak np. dostępne opcje, nie mające na czym pracować (przed wczytaniem obrazka), co w konsekwencji powodowało nieoczekiwane przerwanie pracy programu przy ich wyborze. Te oraz inne błędy zostały jednak poprawione, a ostatnie testy przynoszą wyniki, na podstawie których możemy stwierdzić poprawność działania całej aplikacji w kontekście postawionych wymagań.

VIII. Wdrożenie i raport.

1. Efekty pracy.

Poniżej przedstawiono kilka stereogramów stworzonych za pomocą programu „Stereogramiksy”.





2. Przyszłość.

Projekt może być rozwijany w przyszłości. Istnieje możliwość wyposażenia go w inne algorytmy tworzące obrazy stereoskopowe, ale również poszerzenia o nowe moduły. W najbliższym czasie najlepszym rozwiązaniem byłoby rozbudowanie dodatkowych modułów, jak gra, czy kreator scenariuszy. Chociaż spełniają swoje role, to jednak sam ich projekt jest dosyć ubogi, a obsługa niewygodna. Brakuje takich opcji, jak edytowanie gotowego scenariusza, usuwanie niechcianego obrazka lub hasła, a wszystkie generowane stereogramy stanowiące etapy gry mają białe tło. Dopiero po pełnym dopracowaniu tych elementów można by się zastanawiać, w jaki jeszcze sposób wzbogacić aplikację.