

Tematy projektów z Pythona

Języki Symboliczne 2021, studia stacjonarne

Spis treści

Uwagi ogólne	3
Tematy i opisy projektów	5
Automat biletowy MPK	5
Opis zadania	5
Testy	5
Parkomat	6
Opis zadania	6
Testy	6
Automat sprzedający napoje	8
Opis zadania	8
Testy	8
Program do wyszukiwania połączeń kolejowych	9
Opis zadania	9
Testy	9
Warcaby	10
Opis zadania	10
Testy	10
Cztery w rzędzie (https://en.wikipedia.org/wiki/Connect_Four)	11
Opis zadania	11
Testy	11
Mastermind (https://en.wikipedia.org/wiki/Mastermind_(board_game))	11
Opis zadania	11
Testy	12
Symulacją obsługi klientów przy okienkach z wyliczaniem statystyk	14
Opis zadania	14
Testy	14
Symulator kasjera	16
Opis zadania	16
Testy	16

Program w Pythonie wywołujący funkcję z biblioteki dll wykonującą operacje na rysunku	18
Opis zadania	18
Testy	18
Program w Pythonie wywołujący funkcję z biblioteki dll liczącą iloczyn macierzy	19
Opis zadania	19
Testy	19
Klient bazy danych	21
Opis zadania	21
Testy	21
Saper (https://pl.wikipedia.org/wiki/Saper_(gra_komputerowa))	22
Opis zadania	22
Testy	23
Okręty (https://pl.wikipedia.org/wiki/Okr%C4%99ty)	24
Opis zadania	24
Testy	24
Generator labiryntów	25
Opis zadania	25
Testy	25

Uwagi ogólne

1. Zasady wyboru tematu projektu
 - 1.1. Tematy można wybrać z puli przykładowych tematów zawartych w tym dokumencie.
 - 1.2. Możliwe jest zgłoszenie własnego tematu. Należy przesłać prowadzącemu propozycje opisu i testów (w formie opisanej w dalszej części dokumentu). Własne tematy muszą być zgłoszone i zaakceptowane w jednoznaczny sposób przez prowadzącego najpóźniej do 10 maja 2021. W szczególności stwierdzenie: "temat jest w porządku, ale należy zmienić X" nie jest zaakceptowaniem tematu.
 - 1.3. Opisy projektów i testów mogą ulec niewielkim modyfikacjom mającym na celu usunięcie niejasności lub dodanie autorskich rozwiązań.
 - 1.4. W danej grupie laboratoryjnej możliwa jest realizacja danego tematu niezależnie przez maksymalnie dwie osoby - dlatego o skompletowanie listy zbiorczej tematów dla danej grupy i przesłanie jej za pomocą wiadomości w systemie Moodle proszeni są starostowie danej grupy lub osoby wybrane przez daną grupę. W przypadku braku takiej listy decyzję o przyznaniu lub nie danego tematu (z puli dostępnych poniżej) danej osobie podejmie prowadzący do dnia 12 maja 2020.
2. Zasady opisu i zgłoszenia danego zadania
 - 2.1. Opis zadań i testów powinien być analogiczny do zamieszczonych w poniższym dokumencie
 - 2.2. Wybrany temat (zarówno ten z dostępnej poniżej puli jak i indywidualny) musi być opisany w pliku typu Markdown z rozszerzeniem „.md”.
 - 2.3. Nazwa pliku z opisem powinna mieć format: „opis_XXX.md”, gdzie XXX oznacza „temat_wybranego_projektu”
 - 2.4. Tematy projektów wraz z ich opisem oraz opisem testów należy przesłać w systemie Moodle w zadaniu pt. „Wybór tematu projektu”
 - 2.5. Dla każdego projektu należy stworzyć jego repozytorium (publiczne) w systemie GitHub oraz zamieścić link do niego w przesyłanym opisie.
3. Zasady oddawania projektów
 - 3.1. Ostateczny termin oddania projektu to 9 czerwca 2021r.
 - 3.2. Raport z projektu spakowany wraz ze źródłami (skopiowanymi z GitHub) należy przesłać w systemie Moodle w zadaniu pt. „Projekt z Pythona” od 2 do 9 czerwca 2021r.
4. Wymagania w stosunku do projektów oraz raportu
 - 4.1. Wymagania obowiązkowe:
 - a. Konieczne jest udokumentowanie wymaganych przypadków użycia mieszczących się w ramach opisu zadania.
 - b. Wszystkie dane wprowadzane przez użytkownika powinny być sprawdzane, w razie wpisania niepoprawnych wartości powinna zostać wyświetlona wiadomość informująca użytkownika.
 - c. Do rysowania okienek polecana jest biblioteka tkinter (<https://docs.python.org/3/library/tk.html>).
5. Zawartość raportu oraz sposób oceny projektu
 - 5.1. Raport ma być treściwy i w miarę krótki. Ma zawierać założenia projektowe kodu, ogólny opis kodu, co udało się zrobić, z czym były problemy, dodane elementy specjalne, zauważone problemy z testami.
 - 5.2. Na końcu raportu muszą się znaleźć opisane linki do istotnych fragmentów kodu (w źródłach na GitHub) który obrazuje wymagane w projekcie konstrukcje takie jak:

- a. Lambda-wyrażenia
- b. List comprehensions
- c. Klasy
- d. Wyjątki
- e. Moduły

- Linki mogą zostać stworzone np. tak jak opisano to w:
<https://help.github.com/en/github/managing-your-work-on-github/creating-a-permanent-link-to-a-code-snippet>
- Idea linków do kodu zakłada, że osoba oceniająca projekt szybko się po ich zawartości zorientuje jak zgodny z założeniami jest projekt i ułatwi jego ocenę.

5.3. Punktacja:

- a. 10% - zgodność programu z opisem w temacie zadania
 - Wygląd/interfejs programu może być inny niż w opisie, pod warunkiem, że wszystkie podstawowe funkcjonalności pozostaną zgodne z opisem (na przykład przyciski monet można zastąpić zdjęciami, plansza i pionki w warcabach mogą być graficzne)
 - Dodatkowe funkcjonalności można dodawać wedle uznania, pamiętając jednak, że im większy program tym więcej miejsc w których można popełnić błąd (przykładowo odtwarzanie dźwięku wrzucanej monety, animacja wybuchu podczas zbijania pionka w warcabach).
- b. 20% - poprawność funkcjonalna
 - Seria testów zależna od tematu, podana razem z opisem tematu. Każdy zaliczony test daje $1/N \cdot 20\%$ punktów, gdzie N to liczba testów przewidzianych dla danego tematu.
- c. 20% - poprawne wykorzystanie czterech wybranych z poniższej listy konstrukcji Pythona (każda za 5%):
 - Lambda-wyrażenia (minimum 3 przykłady; np. obsługa kliknięcia w przyciski).
 - List comprehensions lub dictionary comprehensions (min. 3 przykłady w projekcie).
 - Klasy -- wypełnienie przynajmniej jednego z poniższych:
 1. Dziedziczenie: minimum 4 własne klasy, z czego minimum dwie dziedziczą po innych samodzielnie napisanych klasach; minimum dwie nadpisane metody wirtualne.
 2. Podział na klasę odpowiedzialną za interfejs użytkownika i na drugą realizującą funkcjonalność programu. Wszystkie atrybuty obu klas muszą być prywatne.
 - Wyjątki (m.in. walidacja danych wprowadzanych przez użytkownika) - zdefiniowanie własnej klasy wyjątku, rzucanie wyjątku, łapanie wyjątku
 - Własne moduły (podział programu na przynajmniej dwa pliki, np. jeden do obsługi logiki i drugi do obsługi interfejsu).
- d. 10% - wyróżniające elementy, w szczególności:
 - Wykorzystanie zaawansowanych konstrukcji Pythona (np. generatory, metaklasy, dekoratory).
 - Wykorzystanie zaawansowanych algorytmów zapewniających dodatkową funkcjonalność ponad opisane minimum.
- e. 40% - Czytelność i udokumentowanie kodu (Docstrings, komentarze) oraz aktywność na GitHub (ilość i systematyka dokonywania komitów – ogólnie pojęte „development activity”)

Tematy i opisy projektów

1. Automat biletowy MPK

Opis zadania

- Automat przechowuje informacje o monetach/banknotach znajdujących się w nim (1, 2, 5, 10, 20, 50gr, 1, 2, 5, 10, 20, 50zł) [dziedziczenie: można napisać uniwersalną klasę PrzechowywaczMonet po której dziedziczyć będzie automat]
- Okno z listą biletów w różnych cenach (jako przyciski). Wymagane bilety: 20-minutowy, 40-minutowy, 60-minutowy w wariantach normalnym i ulgowym.
- Możliwość wybrania więcej niż jednego rodzaju biletu. Możliwość wprowadzenia liczby biletów.
- Po wybraniu biletu pojawia się okno z listą monet (przyciski) oraz możliwością dodania kolejnego biletu lub liczby biletów.
- Interfejs ma dodatkowo zawierać pole na wybór liczby wrzucanych monet (domyślnie jedna).
- Po wrzuceniu monet, których wartość jest większa lub równa cenie wybranych biletów, automat sprawdza czy może wydać resztę.
 - Brak reszty/może wydać: wyskakuje okienko z informacją o zakupach, wydaje resztę (dolicza wrzucone monety, odlicza wydane jako reszta), wraca do wyboru biletów.
 - Nie może wydać: wyskakuje okienko z napisem "Tylko odliczona kwota" oraz zwraca włożone monety.

Testy

1. Bilet kupiony za odliczoną kwotę - oczekiwany brak reszty.
2. Bilet kupiony płacąc więcej - oczekiwana reszta.
3. Bilet kupiony płacąc więcej, automat nie ma jak wydać reszty - oczekiwana informacja o błędzie oraz zwrócenie takiej samej liczby monet o tych samych nominałach, co wrzucone.
4. Zakup biletu płacąc po 1gr - suma stu monet 1gr ma być równa 1zł (dla floatów suma sto razy 0.01+0.01+...+0.01 nie będzie równa 1.0). Płatności można dokonać za pomocą pętli for w interpreterze.
5. Zakup dwóch różnych biletów naraz - cena powinna być sumą.
6. Dodanie biletu, wrzucenie kilku monet, dodanie drugiego biletu, wrzucenie pozostałych monet, zakup za odliczoną kwotę - oczekiwany brak reszty (wrzucone monety nie zerują się po dodaniu biletu).
7. Próba wrzucenia ujemnej oraz niecałkowitej liczby monet (oczekiwany komunikat o błędzie).

2. Parkomat

Opis zadania

- Parkomat przechowuje informacje o monetach/banknotach znajdujących się w nim (1, 2, 5, 10, 20, 50gr, 1, 2, 5, 10, 20, 50zł)
- Okno z polem tekstowym na numer rejestracyjny pojazdu, aktualną datą (rok, miesiąc, dzień, godzina, minuta), datą wyjazdu z parkingu (rok, miesiąc, dzień, godzina, minuta), przyciskami pozwalającymi na wrzucanie monet (proszę umieścić pole pozwalające wpisać liczbę wrzucanych monet), oraz przyciskiem "Zatwierdź".
- Program powinien zawierać pole pozwalające na przedstawienie aktualnego czasu.
- Zasady strefy parkowania:
 - Strefa płatnego parkowania obowiązuje w godzinach od 8 do 20 od poniedziałku do piątku.
 - Pierwsza godzina płatna 2zł.
 - Druga godzina płatna 4zł.
 - Trzecia i kolejne godziny płatne po 5zł.
 - Czas wychodzący poza obowiązywanie płatnego parkowania przechodzi na kolejny dzień
 - Wykupienie godziny parkowania o 19:20 w piątek pozwala na parkowanie do 8:20 w poniedziałek (koniec o 20:20, wychodzi 20 minut poza płatne parkowanie, przechodzi na kolejny płatny dzień).
- Po każdym wrzuceniu monety termin wyjazdu aktualizuje się zgodnie z całą wrzuconą kwotą.
- Jeśli wrzucone zostało mniej pieniędzy niż potrzeba na opłacenie pełnej godziny, to opłacana jest niepełna godzina:
 - Wrzucenie 1zł pozwala na parkowanie 30 minut,
 - Wrzucenie 5zł pozwala na parkowanie 1 godzinę i 45 minut (2zł na opłacenie pierwszej godziny, zostało 3zł, a potrzeba 4zł na opłacenie kolejnej, co daje 3/4 godziny: 45 minut).
- Po wciśnięciu przycisku "Zatwierdź" wyświetlane jest okno z potwierdzeniem opłacenia parkingu: numer rejestracyjny pojazdu, czas zakupu i termin wyjazdu.
- Numer rejestracyjny może składać się tylko z wielkich liter od A do Z i cyfr.
- W automacie mieści się dowolna liczba banknotów (10, 20, 50zł) i po 200 monet każdego rodzaju. Próba wrzucenia monety ponad limit powoduje wyświetlenie informacji o przepełnieniu parkomatu i prośbę o wrzucenie innego nominału.

Testy

1. Ustaw niepoprawną godzinę. Oczekiwany komunikat o błędzie. Ustawić godzinę na 12:34.
2. Wrzucić 2zł, oczekiwany termin wyjazdu godzinę po aktualnym czasie. Dorzuć 4zł, oczekiwany termin wyjazdu dwie godziny po aktualnym czasie. Dorzuć 5zł, oczekiwany termin wyjazdu trzy godziny po aktualnym czasie. Dorzuć kolejne 5zł, oczekiwany termin wyjazdu cztery godziny po aktualnym czasie.
3. Wrzucić tyle pieniędzy, aby termin wyjazdu przeszedł na kolejny dzień, zgodnie z zasadami -- wrzucić tyle monet aby termin wyjazdu był po godzinie 19:00, dorzucić monetę 5zł,

4. Wrzucić tyle pieniędzy, aby termin wyjazdu przeszedł na kolejny tydzień, zgodnie z zasadami - wrzucić tyle monet aby termin wyjazdu był w piątek po godzinie 19:00, a potem dorzucić monetę 5zł,
5. Wrzucić 1zł, oczekiwany termin wyjazdu pół godziny po aktualnym czasie,
6. Wrzucić 200 monet 1gr, oczekiwany termin wyjazdu godzinę po aktualnym czasie.
7. Wrzucić 201 monet 1gr, oczekiwana informacja o przepełnieniu parkomatu.
8. Wciśnięcie "Zatwierdź" bez wrzucenia monet -- oczekiwana informacja o błędzie.
9. Wciśnięcie "Zatwierdź" bez wpisania numeru rejestracyjnego -- oczekiwana informacja o błędzie. Wciśnięcie "Zatwierdź" po wpisaniu niepoprawnego numeru rejestracyjnego -- oczekiwana informacja o błędzie.

3. Automat sprzedający napoje

Opis zadania

- Automat przechowuje informacje o monetach znajdujących się w nim (1, 2, 5, 10, 20, 50gr, 1, 2, 5zł)
- Automat przechowuje informacje o towarach znajdujących się w nim (przedmioty o numerach od 30 do 50), każdy o określonej cenie w określonej liczbie (domyślnie po 5 sztuk każdego towaru)
- Okno z przyciskami pozwalającymi na wrzucanie monet, polem wyświetlającym kwotę wrzuconych monet, przyciskiem przerywającym transakcję (wyrzuca wrzucone monety), przyciskami 0-9 pozwalającymi wpisać numer wybranego towaru oraz polem wyświetlającym wpisany numer towaru.
- Po wybraniu poprawnego numeru towaru:
 - Jeśli wrzucono za mało monet, wyskakuje okno z informacją o cenie towaru oraz (jeśli towar się skończył) o jego braku w automacie.
 - Jeśli wrzucono monety, których wartość jest większa lub równa cenie wybranego towaru, automat sprawdza czy towar jest dostępny i czy może wydać resztę
 - Brak towaru: wyskakuje okienko z informacją o braku w automacie.
 - Brak reszty/może wydać: wyskakuje okienko z informacją o zakupach, wydaje resztę (dolicza wrzucone monety, odlicza wydane jako reszta, odlicza wydany towar), odejmuje towar.
 - Nie może wydać: wyskakuje okienko z napisem "Tylko odliczona kwota".

Testy

1. Sprawdzenie ceny jednego towaru - oczekiwana informacja o cenie.
2. Wrzucenie odliczonej kwoty, zakup towaru - oczekiwany brak reszty.
3. Wrzucenie większej kwoty, zakup towaru - oczekiwana reszta.
4. Wykupienie całego asortymentu, próba zakupu po wyczerpaniu towaru - oczekiwana informacja o braku.
5. Sprawdzenie ceny towaru o nieprawidłowym numerze (<30 lub >50) - oczekiwana informacja o błędzie.
6. Wrzucenie kilku monet, przerwanie transakcji - oczekiwany zwrot monet.
7. Wrzucenie za małej kwoty, wybranie poprawnego numeru towaru, wrzucenie reszty monet do odliczonej kwoty, ponowne wybranie poprawnego numeru towaru - oczekiwany brak reszty.
8. Zakup towaru płacąc po 1 gr - suma stu monet ma być równa 1zł (dla floatów suma sto razy 0.01+0.01+...+0.01 nie będzie równa 1.0). Płatności można dokonać za pomocą pętli for w interpreterze.

4. Program do wyszukiwania połączeń kolejowych

Opis zadania

- Dwa okna:
 - Pierwsze: ustawianie połączeń (dwie kolumny przycisków opcji (radio button) lub listy rozwijane z nazwami miejscowości), przycisk "połącz", przycisk "rozłącz" oraz macierz istniejących połączeń (macierz sąsiedztwa grafu połączeń). Użytkownik wybiera miejscowości i tworzy między nimi połączenie.
 - Drugie: wyszukiwanie połączenia, dwie listy rozwijane, pierwsza oznaczająca miasto startowe, druga miasto docelowe, przycisk "szukaj", oraz lista przystanków na znalezionej trasie
- Program na początku ma zdefiniowane kilka (10) miast oraz połączeń między nimi (początkowo do każdego miasta da się dojechać, ale nie z każdego bezpośrednio do innego).
- Użytkownik wybiera miasto z którego chce dojechać, oraz miasto docelowe i wciska przycisk "szukaj":
 - Połączenie znalezione: wpisuje do listy kolejne przystanki, np: "Kraków -> Katowice", "Katowice -> Wrocław", "Wrocław -> Ankh-Morpork"
 - Połączenia brak: wyskakuje okienko z informacją "Brak połączenia na tej trasie"
- Program powinien realizować wyszukiwanie połączeń za pomocą algorytmu BFS. Graf połączeń powinien mieć dwie możliwe reprezentacje (do wyboru w pierwszym oknie): macierz sąsiedztwa albo listy sąsiedztwa, realizowane jako dwie klasy dziedziczące po klasie AbstractGraph definiującej niezbędny interfejs (pobranie listy wierzchołków, pobranie krawędzi incydentnych z wierzchołkiem, pobranie wierzchołków łączonych przez krawędź itp.). Zmiana reprezentacji powinna być możliwa w dowolnym momencie działania programu. Implementacja algorytmu BFS powinna być jedna i działać na obu możliwych reprezentacjach grafu (ma wykorzystywać wyłącznie metody klasy AbstractGraph definiujące interfejs).

Testy

1. Wyszukanie połączenia bezpośredniego między dwoma miastami.
2. Wyszukanie połączenia między dwoma miastami z przynajmniej dwoma przystankami pośrednimi.
3. Dodanie połączenia bezpośredniego między dwoma miastami z punktu b.
- oczekiwana informacja o połączeniu bezpośrednim.
4. Usunięcie połączeń bezpośrednich między miastami z punktu a.,
wyszukanie połączenia - oczekiwana informacja o przesiadce.
5. Usunięcie wszystkich połączeń do danego miasta - oczekiwana informacja o braku połączenia.
6. Wykonanie testu b przy drugiej reprezentacji grafu.
7. Wykonanie testu c przy drugiej reprezentacji grafu.

5. Warcaby

Opis zadania

- Okno z siatką przycisków 8x8 oraz przyciskiem do resetowania gry.
- Przyciski reprezentują pola planszy do gry w warcaby. Pola puste - przyciski bez tekstu. Pola z pionkami gracza 1 - przycisk z tekstem "C". Pola z pionkami gracza 2 - przycisk z tekstem "B". Damki oznaczane są dodatkową literą d ("Cd", "Bd").
- Nad planszą wyświetlna jest informacja "Tura gracza 1" lub "Tura gracza 2".
- Gracz wybiera pionka (tekst pola zmienia się z "C" na "[C]" lub z "B" na "[B]"), a potem pole na które chce wykonać ruch. Jeśli ruch jest dozwolony, pionek jest przestawiany. Jeśli nie, to wyświetlany jest komunikat "ruch niedozwolony".
- Zasady jak w warcabach (<https://pl.wikipedia.org/wiki/Warcaby>, dowolny wariant). Zwykłe pionki i damki mają być obiektami dwóch różnych klas dziedziczących po klasie Pionek.
- Gdy gra się kończy, wyświetlane jest okienko z napisem "Wygrał gracz 1" lub "Wygrał gracz 2", zależnie kto wygrał grę. Możliwe jest zresetowanie planszy bez zamknięcia głównego okna.

Testy

1. Wykonanie po dwa ruchy przez każdego z graczy.
2. Niepowodzenie błędного ruchu pionkiem.
3. Wykonanie bicia pojedynczego pionka.
4. Wykonanie bicia przynajmniej dwóch pionków.
5. Zamiana pionka w damkę.
6. Bicie damką.
7. Wygrana gracza grającego czarnymi pionkami.
8. Rozpoczęcie nowej gry po zwycięstwie jednego z graczy.

Wskazane jest przygotowanie specjalnych początkowych rozstawień pionków dla testów d, e, f, g, h.

6. Cztery w rzędzie

(https://en.wikipedia.org/wiki/Connect_Four)

Opis zadania

- Okno wyświetlające siatkę 7 kolumn x 6 wierszy, przycisk nad każdą kolumną, informację “Tura gracza 1” lub “Tura gracza 2”, przycisk do resetowania gry oraz rozwijalną listę wyboru reguł gry.
- Początkowo pola siatki są puste.
- Gracze na zmianę wrzucają monety do wybranych przez siebie kolumn.
- Pola w których jest moneta gracza 1 są czerwone, pola z monetami gracza 2 są żółte (tkinter, Canvas, <http://stackoverflow.com/a/12254442>).
- Gracze wybierają kolumnę klikając przycisk nad nią.
- Wygrywa gracz który pierwszy ustawia cztery monety w linii (poziomo, pionowo lub po skosie).
- Gdy gra się kończy, wyświetlane jest okienko z napisem “Wygrał gracz 1” lub “Wygrał gracz 2”, zależnie kto wygrał grę. Możliwe jest zresetowanie planszy bez zamknięcia głównego okna.
- Reprezentacja reguł gry ma być realizowana poprzez hierarchię klas. Klasa bazowa definiuje między innymi funkcję wirtualną ktoWygral() nadpisywana w klasach pochodnych. Realizowane powinny być przynajmniej dwa zestawy reguł, jako dwie klasy pochodne.

Testy

1. Wykonanie po dwa ruchy przez każdego z graczy - monety spadają na dół pola gry lub zatrzymują się na już wrzuconym żetonie.
2. Ułożenie pionowej linii monet przez jednego gracza - oczekiwana informacja o jego wygranej.
3. Ułożenie poziomej linii monet przez drugiego gracza - oczekiwana informacja o jego wygranej.
4. Ułożenie skośnej linii przez dowolnego gracza - oczekiwana informacja o jego wygranej.
5. Zapełnienie pola gry tak, że żaden gracz nie ułożył linii - oczekiwana informacja o remisie.
6. Ułożenie linii dłuższej niż 4 przez jednego z graczy - oczekiwana informacja o jego wygranej.
[c][c][c][][c][c][c]
[ż][ż][ż][][ż][ż][ż] <- w kolejnym ruchu gracz żółty wrzuci monetę w środkową kolumnę.
7. Próba wrzucenia monety do zapełnionej kolumny - oczekiwana informacja o błędzie.

7. Mastermind

([https://en.wikipedia.org/wiki/Mastermind_\(board_game\)](https://en.wikipedia.org/wiki/Mastermind_(board_game)))

Opis zadania

- Okno z polem tekstowym na 4 cyfry, listą odpowiedzi, przyciskiem "Sprawdź", przyciskiem "Oszust!" oraz przyciskiem "Reset".
- Po rozpoczęciu gry generowana jest losowa liczba (kod) złożona z czterech cyfr od 1 do 6 włącznie (1111, 1112, 1113, ..., 3455, 3456, 3461, 3462, ..., 6665, 6666).
- Gracz wpisuje cztery cyfry od 1 do 6 do pola tekstowego i naciska przycisk "Sprawdź".
- Do pola odpowiedzi dopisywana jest odpowiedź zawierająca: liczbę wpisaną przez gracza, liczbę cyfr na poprawnych pozycjach oraz liczbę cyfr występujących w kodzie, ale na złych pozycjach.
- Jeśli gracz wpisał liczbę będącą kodem, wyświetlane jest okno z napisem "Wygrana".
- Jeśli gracz po 12 próbach nie odgadł kodu, wyświetlane jest okno z napisem "Przegrana".
- Logika gry powinna być realizowana przez osobną klasę, dziedziczącą po klasie RegulyGry. Z klasy RegulyGry powinna być również wydziedziczona druga klasa, generująca niepoprawne odpowiedzi. Wybór zestawu reguł powinien być dokonywany losowo przed każdą grą.
- Jeśli gracz naciągnął przycisk "Oszust!" przy poprawnych regułach gry, program powinien wyświetlić okno z napisem "Tere fere." oraz wylosowanym kodem.
- Jeśli gracz naciągnął przycisk "Oszust!" przy niepoprawnych regułach gry, program powinien wyświetlić okno z napisem "Złapałeś/łaś mnie!"

Testy

1. Wyświetlenie (wypisanie w konsoli) wylosowanego kodu, wpisanie odpowiedzi z błędymi cyframi - oczekiwana informacja o braku poprawnych trafień.
2. Wyświetlenie wylosowanego kodu, wpisanie odpowiedzi z poprawnymi cyframi w złych miejscach - oczekiwana informacja o niepoprawnym położeniu.
3. Wyświetlenie wylosowanego kodu, wpisanie odpowiedzi z dwoma poprawnymi cyframi w dobrych miejscach i dwoma poprawnymi w złych miejscach - oczekiwana informacja o dwóch trafieniach i dwóch złych pozycjach.
4. Wyświetlenie wylosowanego kodu, wpisanie poprawnej odpowiedzi - oczekiwana informacja o wygranej.
5. Wpisanie 12 razy niepoprawnego kodu - oczekiwana informacja o przegranej
6. Próba wpisania niepoprawnego kodu do pola odpowiedzi (mniej lub więcej niż 4 znaki, znaki nie będące cyframi od 1 do 6) - oczekiwane nieuznanie kodu (gracz nie traci tury).
7. Wciśnięcie przycisku "Oszust" przy poprawnych zasadach gry - oczekiwana informacja "tere fere".
8. Wciśnięcie przycisku "Oszust" przy niepoprawnych zasadach gry - oczekiwana informacja o oszukiwaniu przez komputer.

9. Wpisanie 10 kodów, resetowanie gry, wpisanie 5 kodów - oczekiwane normalne działanie gry (czy licznik tur resetuje się po wciśnięciu "Reset").

8. Symulacją obsługi klientów przy okienkach z wyliczaniem statystyk

Opis zadania

- Trzy okienka (na przykład w urzędzie) obsługują klientów przychodzących w jednej z dwóch spraw (A, B).
- Okienka reprezentowane są przez obszary w dole okna programu z wyborem (checkbox) spraw jakie są tam załatwiane i przyciskiem "Następny".
- Nowe osoby chcące coś załatwić reprezentowane są przez wciśnięcia jednego z czterech przycisków na górze okna programu ("A", "B", "VIP A", "VIP B"). Obok przycisków napisana jest liczba czekających klientów z daną sprawą.
- Program automatycznie przydziela osoby do okienek w momencie naciśnięcia przycisku "Następny". Osoby są przyjmowane w kolejności przyjścia, najpierw z kolejki VIP, a następnie (gdy kolejka VIP jest pusta) ze zwykłych kolejek. Jeśli zwykłe kolejki też są puste, to następny pasujący klient który przyjdzie przyjmowany jest bez czekania.
- Jeśli klient VIP czeka dłużej niż 10 sekund, powinno się wyświetlić okno z tekstem "To skandaliczne!".
- Klienci są reprezentowani przez klasy KlientZwykły i KlientVIP dziedziczące po klasie Klient. Posiadają one metody wejście, wyjście i tick odpowiednio wywoływane przy wejściu klienta, jego wyjściu i co sekundę, rejestrujące odpowiednie czasy i reagujące na zdarzenia.
- Program posiada również w głównym oknie przycisk "Zakończ" wyświetlający statystyki (średni czas oczekiwania w poszczególnych okienkach i w ogóle z rozbiciem na sprawy i bez) jak poniżej i resetujący stan. Wyświetlone mają być dwie tabele, osobno dla zwykłych klientów i klientów VIP.

---	Okienko 1	Okienko 2	Okienko 3	Łącznie
Sprawa A				
Sprawa B				
Łącznie				

Testy

1. Ustawienie okienka pierwszego na obsługę spraw A, drugiego na obsługę spraw B, a trzeciego na obsługę obu typów spraw (dotyczy wszystkich podpunktów o ile nie wyszczególniono inaczej). Wpuszczenie klienta ze sprawą A, naciśnięcie obsługi przy okienku drugim, naciśnięcie obsługi przy okienku pierwszym, wpuszczenie klienta ze sprawą B (powinien od razu być załatwiony w drugim okienku), zakończenie.
2. Wpuszczenie klienta A, wpuszczenie klienta VIP A, następny w okienku pierwszym, następny w okienku trzecim. Czasy obsługi powinny wskazywać na to, że klient VIP był obsłużony jako pierwszy.

3. Wpuszczenie klienta VIP A, wpuszczenie klienta B, następny w okienku trzecim, następny w okienku drugim, zakończenie.
4. Wpuszczenie klienta VIP B, wpuszczenie klienta A, następny w okienku trzecim, następny w okienku pierwszym, zakończenie.
5. Wpuszczenie klienta A, zakończenie (podsumowanie powinno wskazywać czas od wejścia klienta do zakończenia).
6. Wpuszczenie klienta A, wpuszczenie klienta VIP B, następny w okienku pierwszym, oczekanie do momentu wyświetlenia komunikatu o zbyt wolnej obsłudze.
7. Wpuszczenie po czterech klientów każdego rodzaju i obsługa ich wszystkich w okienkach pierwszym i drugim. Zakończenie.

9. Symulator kasjera

Opis zadania

- Okno podzielone jest na dwie części:
 - Lewa: pusta, pojawiają się tam towary do skasowania
 - Prawa: przyciski od 0 do 9, przycisk "backspace", przycisk "Wyczyść", przycisk "Zważ" oraz pole tekstowe - wciskanie przycisków cyfr powoduje dopisywanie ich do pola, backspace wymazuje ostatnio wpisaną cyfrę, przycisk "Wyczyść" czyści pole tekstowe.
Jeśli zawartość pola tekstowego to 1 i wciśnięty został przycisk cyfry, pole tekstowe jest czyszczone i dopisywana jest tam wybrana cyfra.
- Na początku po lewej stronie znajduje się przycisk "Następny klient", wciśnięcie go rozpoczyna grę (zapisanie aktualnego czasu i wyzerowanie licznika towarów)
- W losowym miejscu lewej strony okna pojawia się przycisk z:
 - nazwą towaru i jego liczebnością, np. "Arбуз x10". Wciśnięcie przycisku powoduje zmniejszenie liczebności towaru o tyle, ile wpisane jest w polu tekstowym po prawej stronie, a pole to jest czyszczone (zawartość ustawiana na 1) lub
 - nazwą towaru i napisem "?kg" oznaczającym, że jest to towar do zważenia. Jego kasowanie polega na kliknięciu przycisku z towarem, następnie kliknięciu przycisku "Zważ", a następnie ponownym kliknięciu przycisku z towarem. Po naciśnięciu przycisku "Zważ" etykieta towaru powinna się zmienić (ma być podana losowa waga od 0.05 do 2 kg).
- Jeśli wartość pola tekstowego po prawej stronie przewyższa liczebność towaru, to gracz przegrywa i wyświetlane jest okno informujące o tym.
- Po skasowaniu towaru (liczebność spadła do 0), do licznika towarów dodawana jest oryginalna liczebność towaru, oraz generowany jest kolejny towar.
- Generowane jest od 10 do 20 towarów, połowa z nich na sztuki. Liczba sztuk ma wynosić od 1 do 50 (losowo). Prawdopodobieństwo wylosowania liczebności 1 (pojedynczy artykuł) ma wynosić 50%.
- Po skasowaniu wszystkich towarów wyświetlany jest średni czas kasowania jednego przedmiotu (twarz w liczebności 10 liczy się za 10 przedmiotów).
- Nazwy towarów mają być losowane z minimum dwudziestoelementowej listy.
- Towary mają być reprezentowane przez obiekty TowarNaSztuki i TowarNaWagę dziedziczących po klasie Towar. W obiektach ma być przechowywana nazwa towaru, liczba sztuk lub waga, czas pojawiienia się w lewej części okna i czas skasowania.

Testy

1. Skasowanie towaru na sztuki po klikając na niego kilka razy.
2. Skasowanie towaru na sztuki wpisując jego liczbność i klikając raz. Wymagane jest resetowanie pola do wartości 1.
3. Próba skasowania towaru na sztuki wpisując zbyt dużą liczbność (oczekiwana informacja o przegranej).
4. Próba zważenia towaru na sztuki (oczekiwana informacja o przegranej).

5. Próba skasowania towaru na wagę jakby był towarem na sztuki (oczekiwana informacja o przegranej).
6. Skasowanie wszystkich towarów (oczekiwane okno z podsumowaniem symulacji).
7. Pokazanie, że liczebność 1 występuje odpowiednio często.

10. Program w Pythonie wywołujący funkcję z biblioteki dll wykonującą operacje na rysunku

Opis zadania

- Program operuje na rysunkach w odcieniach szarości (0-255) przy użyciu podanej maski (różne filtry).
- Okno z polem wyświetlającym rysunek, przyciskiem pozwalającym wybrać rysunek (wczytywanie z pliku), siatką 3x3 pól tekstowych (maska) umożliwiających wprowadzenie maski, oraz przyciskiem "Filtruj".
- Wykorzystywane są toroidalne warunki brzegowe (np. prawymi sąsiadami pikseli na prawym brzegu są piksele z lewego brzegu).
- Użytkownik wczytuje rysunek, ustawia maskę i wciska przycisk "Filtruj".
- Rysunek przesyłany jest do funkcji z pliku .dll, która stosuje podaną maskę na rysunku.
- Po zakończeniu działania funkcji, rysunek w programie powinien przedstawać efekt działania filtru.
- Ponowne wcisnięcie przycisku "Filtruj" powinno działać na zmodyfikowanym rysunku.
- Do wywoływania funkcji z dll należy stosować bibliotekę ctypes (<https://docs.python.org/3.6/library/ctypes.html>).
- Do obsługi obrazów można stosować bibliotekę pillow (<https://python-pillow.org/>).

Testy

1. Wczytanie i wyświetlenie rysunku.
2. Filtrowanie z maską $[[0, 0, 0], [0, 1, 0], [0, 0, 0]]$ z wykorzystaniem dll.
3. Filtrowanie z maską $[[1, 0, -1], [2, 0, -2], [1, 0, -1]]$ (filtr Sobela), z wykorzystaniem dll.
4. Filtrowanie z maską $[[1/9, 1/9, 1/9], [1/9, 1/9, 1/9], [1/9, 1/9, 1/9]]$ (rozmycie), z wykorzystaniem dll.
5. Wczytanie innego rysunku bez zamknięcia programu.
6. Próba wczytania pliku dll (oczekiwany komunikat o błędzie, możliwy ma być wybór innego pliku bez ponownego uruchamiania programu).
7. Wykonanie filtrów b, c i d kolejno na tym samym rysunku (rysunek \rightarrow filtr b \rightarrow rysunek z filtrem b \rightarrow filtr c \rightarrow rysunek z filtrami b i c \rightarrow filtr d \rightarrow rysunek z filtrami b, c, d)

11. Program w Pythonie wywołujący funkcję z biblioteki dll liczącą iloczyn macierzy

Opis zadania

- Okno z przyciskiem “Uruchom obliczenia”, polem z informacjami o ostatnim uruchomieniu (czas obliczeń w Pythonie, czas obliczeń w C/C++, czas potrzebny na konwersję danych), oraz polami pozwalającymi wybrać:
 - Liczbę kolumn macierzy,
 - Liczbę wierszy macierzy,
 - Zakres generowanych wartości w macierzach (od, do),
 - Rodzaj generowanych wartości (liczby całkowite/rzeczywiste),
 - Liczba powtórzeń mnożenia (do uśrednienia czasu obliczeń).
- Powinna dodatkowo istnieć możliwość ręcznego wprowadzenia niewielkich (do trzech wierszy i kolumn) macierzy które mają być przemnożone. Wprowadzanie tych macierzy można zrealizować przez tablice pól tekstowych do wprowadzania elementów macierzy.
- Po wciśnięciu przycisku “Uruchom obliczenia” generowane są dwie macierze o podanych przez użytkownika parametrach, które są następnie mnożone w Pythonie oraz w C/C++ (kod mnożący wywoływany z pliku .dll). Jeśli macierzy nie da się przemnożyć, ma zostać wyświetlony odpowiedni komunikat w oknie dialogowym (proszę użyć mechanizmu wyjątków).
- Implementacje w Pythonie i w C lub C++ mają być realizowane w dwóch klasach (MnozenieMacierzyCpp, MnozenieMacierzyPython) dziedziczących po klasie MnozenieMacierzy z metodą mnoz.
- Zapisywany jest czas obliczeń w pierwszym i drugim języku oraz czas przygotowania danych do przesłania do C/C++.
- Wyniki wpisywane są do pola z informacjami.
- Do wywoływania funkcji z dll należy stosować bibliotekę ctypes (<https://docs.python.org/3.6/library/ctypes.html>).

Testy

1. Wykonanie mnożenia (w Pythonie i przez dll):

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

2. Wykonanie mnożenia (w Pythonie i przez dll):

$$\begin{bmatrix} 1 & 2 & 1 \\ 4 & -2 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 2 & 0 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ -4 & 4 \end{bmatrix}$$

3. Wykonanie mnożenia (w Pythonie i przez dll):

$$\begin{bmatrix} 1 & 2 \\ 4 & -2 \\ 0 & 3 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & -3 \\ 2 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 4 & -1 & -3 \\ -4 & 6 & -12 \\ 6 & -3 & 0 \end{bmatrix}$$

4. Próba wykonania mnożenia macierzy 2x3 przez macierz 1x2 (oczekiwany komunikat o błędzie i możliwość kontynuowania pracy z programem bez ponownego uruchamiania).
5. Uzyskanie porównania czasów dla mnożenia macierzy 1x1, 10x10 i 50x50 (odpowiednio 100000, 1000 i 50 powtórzeń, liczby całkowite).
6. Uzyskanie porównania czasów dla mnożenia macierzy 200x200 (3 powtórzenia, liczby rzeczywiste).

12. Klient bazy danych

Opis zadania

- Program udający klienta bazy danych, pozwalający na utworzenie/usunięcie tabeli, dodawanie/usuwanie wpisów w niej, oraz wyszukanie danych.
- Możliwe utworzenie tabeli o dowolnej liczbie kolumn, każda kolumna zawierająca dane typu *liczba całkowita* albo *liczba rzeczywista* albo *tekst*.
- Typ *liczba całkowita* pozwala na przechowywanie liczb całkowitych (pythonowy *int*).
- Typ *liczba rzeczywista* pozwala na przechowywanie liczb rzeczywistych (pythonowy *float*).
- Typ *tekst* pozwala na przechowywanie dowolnego tekstu (domyślny typ tekstowy pythona).
- Nazwy tabel i kolumn nie mogą być puste (muszą zawierać przynajmniej jeden znak drukowalny).
- Główne okno powinno powinno posiadać listę tabel znajdujących się w bazie danych, listę wpisów w zaznaczonej tabeli (lista lub dowolny **czytelny** sposób prezentacji danych), oraz przyciski "Dodaj tabelę", "Usuń tabelę", "Dodaj wiersz", "Usuń wiersz" i "Wyszukaj w tabeli".
- Okno dodawania tabeli powinno pozwalać na wpisanie nazwy tabeli oraz dodanie kolumn (nazwa i typ).
- Okno dodawania wiersza powinno wyświetlać nazwy kolumn i pola tekstowe służące wpisaniu danych. Walidacja typów wpisanych danych powinna odbywać się przy próbie dodania wiersza (np. Jeśli pole tekstowe kolumny typu *liczba całkowita* zawiera tekst nie rzutujący się na liczbę całkowitą, to powinien zostać wyświetlony błąd - okienko, znaczek na polu tekstowym lub inne).
- Przed usunięciem tabeli/wiersza użytkownik powinien być proszony o potwierdzenie.
- Okno wyszukiwania powinno zawierać pole tekstowe, do którego wpisywany jest kod lambda-wyrażenia zwracającego prawdę lub fałsz i przyjmującego jeden argument - słownik *kolumna:wartość*. Lambda ta jest wywoływaną dla każdego wiersza wybranej tabeli, wyświetlane są wiersze dla których lambda zwróci prawdę. Przykładowo: wpisanie "lambda row: row['ID']>5 or row['ocena']<3" zwróci wszystkie wiersze których kolumna "ID" ma wartość większą od 5 lub kolumna "ocena" ma wartość mniejszą od 3.
- Wynik wyszukiwania powinien być wyświetlony w oknie wyszukiwania (np. po utracie focusa w polu tekstowym, po wcisnięciu przycisku wyszukiwania, po każdej edycji),
- **Na wyższą ocenę:** Po zamknięciu programu dane (tabele i ich zawartość) powinny być zapisywane na dysk, a po jego uruchomieniu wczytywane.

Testy

1. Utworzenie tabeli "test1" z kolumnami liczbową "ID" (typ *int*), dwoma tekstowymi "imię" oraz "nazwisko" oraz liczbową "wzrost" (typ *float*).
2. Dodanie wiersza do tabeli "test1" z danymi "1", "Roch", "Przybipięt", "1.50" - oczekiwane powodzenie.
3. Dodanie wiersza do tabeli "test1" z danymi "2", "Ziemniaczysław", "Bulwiasty", "1.91" - oczekiwane powodzenie.

4. Dodanie wiersza do tabeli "test1" z danymi "cztery", "bla", "bla", "-90" - oczekiwane niepowodzenie (dane tekstowe w polu liczbowym).
 5. Dodanie wiersza do tabeli "test1" z danymi "3.14", "pi", "ludolfina", "314e-2" - oczekiwane niepowodzenie (liczba rzeczywista w kolumnie z liczbą całkowitą).
 6. Wyświetlenie zawartości tabeli "test1".
 7. Dodanie trzech kolejnych wierszy do tabeli "test1" i usunięcie dwóch wierszy z niej (pierwszego i środkowego), w obu przypadkach najpierw anulowanie operacji, a potem jej akceptacja.
 8. Utworzenie tabeli "test2" z kolumnami "reserved" typu *string* oraz "kolor" typu *liczba całkowita*.
 9. Dodanie wiersza do tabeli "test2" z danymi (puste pole), "1337" - oczekiwane powodzenie.
 10. Dodanie wiersza do tabeli "test2" z danymi "bla", "1939b" - oczekiwane niepowodzenie (tekst w polu typu *liczba całkowita*).
 11. Usunięcie tabeli "test2", najpierw anulowanie operacji, a potem jej akceptacja.
 12. Próba utworzenia tabeli bez nazwy - oczekiwane niepowodzenie.
 13. Próba utworzenia tabeli o nazwie " " (spacja) - oczekiwane niepowodzenie.
 14. Próba utworzenia tabeli z kolumną bez nazwy - oczekiwane niepowodzenie.
 15. Próba utworzenia tabeli z kolumną o nazwie " " (cztery spacje) - oczekiwane niepowodzenie,
16. Wypełnij tabelę "test1" danymi testowymi (kolejne wartości "ID", "wzrost" między 1.0 i 2.0 , wyszukaj wiersze dla których "wzrost" ma wartość podaną przez prowadzącego oraz "ID" jest liczbą parzystą lub nieparzystą (zależnie od woli prowadzącego)).

13. Saper

([https://pl.wikipedia.org/wiki/Saper_\(gra_komputerowa\)](https://pl.wikipedia.org/wiki/Saper_(gra_komputerowa)))

Opis zadania

- Główne okno zawiera dwa pola tekstowe do wprowadzania rozmiaru planszy ($n \times n$ pól), planszę o wymiarach $n \times n$ pól (np. siatka przycisków), pole tekstowe na wprowadzenie liczby min na planszy, liczbę oznaczonych pól, liczbę min na planszy, oraz przycisk rozpoczęcia nowej gry.
- Wprowadzenie mniejszego rozmiaru planszy niż 2×2 lub większego niż 15×15 , liczby min mniejszej niż 0 lub większej niż m^2 powoduje wyświetlenie komunikatu o błędzie. Nie można rozpocząć gry dopóki te parametry nie są poprawne. Walidacja danych powinna wykorzystywać mechanizm wyjątków.
- Na początku gry na losowych polach umieszczane jest tyle min ile wskazano w polu tekstowym (każde możliwe rozłożenie min jest równie prawdopodobne).
- Po kliknięciu lewym przyciskiem na pole:
 - Jeśli jest tam mina, wyświetlana jest wiadomość o przegranej i gra się kończy,
 - Jeśli w sąsiedztwie pola są miny, na przycisku wyświetlana jest ich liczba a pole dezaktywuje się,
 - W przeciwnym razie sąsiednie pola są sprawdzane tak jakby zostały kliknięte a pole dezaktywuje się.
- Po kliknięciu prawym przyciskiem pole może zostać oznaczone "tu jest mina", po ponownym kliknięciu oznaczenie zmienia się na "tu może być mina", a po kolejnym kliknięciu oznaczenie znika.
- Gra kończy się po kliknięciu wszystkich pól bez min, lub oznaczeniu "tu jest mina" wszystkich pól z minami (i żadnych innych).
- Po naciśnięciu kolejno klawiszy x, y, z, z, y, pola pod którymi są miny stają się ciemniejsze ([https://en.wikipedia.org/wiki/Xzzy_\(computing\)#Other_computer_games_and_media](https://en.wikipedia.org/wiki/Xzzy_(computing)#Other_computer_games_and_media)).

Testy

1. Próba rozpoczęcia gry z rozmiarem planszy i liczbą min: (1 na 1; 1), (5 na 1; 2), (4 na 1; 2), (20 na 500; 12), (5 na 6; -4), (3 na 3; 10), (1 na 10; 5) - oczekiwane komunikaty o błędzie. Wprowadzenie rozmiarów planszy 8 na 8 i liczby min równej 12 na potrzeby kolejnych testów.
2. Kliknięcie pola, wyświetla się liczba min w sąsiedztwie pola,
3. Kliknięcie pola, wyświetla się mina, gra się kończy,
4. Kliknięcie pola, brak min w sąsiedztwie - oczekiwane automatyczne sprawdzenie sąsiadów aż do wyznaczenia obszaru wyznaczonego przez pola sąsiadujące z minami lub krawędzie planszy,
5. Oznaczenie pola jako "tu jest mina" - licznik oznaczonych powinien wzrosnąć o 1,
6. Oznaczenie innego pola jako "tu może być mina",
7. Oznaczenie pola, odznaczenie go, ponowne oznaczenie i ponowne odznaczenie - licznik oznaczonych powinien się odpowiednio aktualizować,
8. Wygranie gry przez kliknięcie wszystkich pól bez min,

9. Wygranie gry przez oznaczenie wszystkich pól z minami (można skorzystać z kodu xyzzy),
10. Próba oznaczenia sprawdzonego pola - oczekiwane niepowodzenie,
11. Sprawdzenie kilku pól bez min, oznaczenie pól “tu jest mina”, rozpoczęcie nowej gry - licznik min powinien się zaktualizować, a pola zresetować.
12. Wpisanie kodu xyzzy, zresetowanie gry - wszystkie pola powinny odzyskać standardowy kolor.

14. Okręty (<https://pl.wikipedia.org/wiki/Okr%C4%99ty>)

Opis zadania

- Okno z dwoma planszami 10x10 pól (np. siatki przycisków) oraz przyciskiem rozpoczęcia gry i przyciskiem reset.
- Na początku graczy rozmieszcza okręty (1x czteromasztowiec, 2x trójmasztowiec, 3x dwumasztowiec, 4x jednomasztowiec).
- Po rozmieszczeniu okrętów przez gracza i wciśnięciu przycisku nowej gry przeciwnik komputerowy losowo rozmieszcza swoje okręty.
- Okręty nie mogą się dotykać ani bokami ani rogami.
- Po rozmieszczeniu okrętów przez obu graczy jeden z nich wykonuje pierwszy ruch (losowo gracz lub komputer).
- Wybór celu przez gracza następuje przez kliknięcie pola, w razie trafienia przycisk staje się czerwony, w przeciwnym razie niebieski (nie można strzelić dwa razy w to samo pole).
- Komputer strzela w losowe, nie wybrane wcześniej pole. Po trafieniu próba znalezienia orientacji statku i zestrzelenie go do końca.
- Gra kończy się gdy któryś gracz straci ostatni okręt, wyświetlane jest okno z informacją o zwycięzcy (np. "Wygrana!", "Przegrana!").
- Opcjonalnie: bardziej zaawansowana sztuczna inteligencja omijająca pola na których na pewno nie może znaleźć się okręt gracza.

Testy

1. Próba niepoprawnego ustawienia okrętu (stykanie się bokami lub rogami). Oczekiwana informacja o błędzie
2. Poprawne rozmieszczenie wszystkich okrętów przez gracza i wciśnięcie przycisku rozpoczęcia gry.
3. Strzelenie w puste pole.
4. Trafienie w okręt przeciwnika.
5. Próba zestrzelenia swojego okrętu - oczekiwane niepowodzenie.
6. Próba ponownego strzelenia w puste pole - oczekiwane niepowodzenie.
7. Próba ponownego strzelenia w okręt przeciwnika - oczekiwane niepowodzenie.
8. Rozmieszczenie części okrętów, wciśnięcie przycisku reset - oczekiwany reset plansz.
9. Poprawne rozmieszczenie wszystkich okrętów, oddanie kilku strzałów, rozpoczęcie nowej gry, ponowne poprawne rozmieszczenie okrętów, oddanie strzałów w te same pola.
10. Wygranie gry (np. Przez pokazanie okrętów przeciwnika). Rozpoczęcie nowej gry bez ponownego uruchamiania programu.
11. Przegranie gry (np. Przez aktywację super-instynktu gracza komputera). Rozpoczęcie nowej gry bez ponownego uruchamiania programu.

15. Generator labiryntów

Opis zadania

- Główne okno programu zawiera kontrolki pozwalające na wybór wielkości labiryntu (liczba pól N na M; para liczb całkowitych nie większych niż 30), wizualizację labiryntu (na przykład jako siatkę kolorowych przycisków) oraz przycisk "generuj".
- Labirynt składa się z pól będących *korytarzem* lub *ścianą*.
- Użytkownik wybiera dwa różne pola będące wejściem i wyjściem. Pola te traktowane będą jak korytarz. Po naciśnięciu przycisku "generuj" następuje generowanie losowego labiryntu.
- Dla każdej pary pól będących korytarzem powinna istnieć ścieżka je łącząca (brak pól odłączonych od reszty labiryntu). Przechodzenie możliwe jest tylko na pola będące korytarzem które sąsiadują krawędzią z danym polem.
- Wygenerowany labirynt powinien posiadać ścieżkę od wejścia do wyjścia, która nie będzie linią prostą (poziomą lub pionową) i która powinna być zaznaczona na wizualizacji.
- Przechowywana jest lista punktów pośrednich ścieżki prowadzącej od wejścia do wyjścia.
- Po wybraniu dowolnego pola będącego korytarzem, dodawane jest ono na koniec listy punktów pośrednich. Następnie powinna zostać znaleziona i zaznaczona najkrótsza ścieżka prowadząca z wejścia do wyjścia przez wszystkie punkty pośrednie.
- Wybranie pola będącego punktem pośrednim powoduje usunięcie danego punktu pośredniego.
- Labirynt nie może posiadać żadnego 'pokoju': obszar 2 na 2 pola korytarza (lub większy).
- Labirynt nie może posiadać żadnych obszarów 3 na 3 pola ściany (lub większych).

Testy

1. Wygenerowanie labiryntu o wymiarach 10 na 12 pól z wejściem i wyjściem na przeciwnych krawędziach.
2. Wygenerowanie labiryntu o wymiarach 20 na 10 pól z wejściem i wyjściem cztery pola od przeciwnych, krótszych krawędzi.
3. Próba wygenerowania labiryntu o wymiarach 10 na 10 z wejściem i wyjściem w jednym polu - oczekiwana informacja o błędzie.
4. Próba wygenerowania labiryntu o wymiarach 10 na 10 z wejściem i wyjściem koło siebie - oczekiwana informacja o błędzie, ścieżka jest linią prostą.
5. Próba wygenerowania labiryntu o wymiarach 10 na 10 z wejściem i wyjściem między którymi jest 1, 2 lub 3 pola odstępu - oczekiwany labirynt bez ścieżki będącej linią prostą.
6. Próba wygenerowania labiryntu którego przynajmniej jeden z wymiarów wynosi 0 lub jest liczbą ujemną - oczekiwana informacja o błędzie.
7. Próba wygenerowania za dużego labiryntu - oczekiwana informacja o błędzie.

8. Wygenerowanie labiryntu o wymiarach 13 na 17, wejście i wyjście w dowolnych miejscach, wyszukanie ścieżki przez punkt pośredni wskazany przez prowadzącego. Wygenerowanie labiryntu o wymiarach 13 na 17, dodanie dwóch punktów pośrednich wskazanych przez prowadzącego, usunięcie pierwszego, dodanie kolejnego. Oczekiwane znalezienie właściwej ścieżki.
9. Próba wyszukania ścieżki przez pole będące ścianą - oczekiwane niepowodzenie przy próbie dodania punktu pośredniego