**Dokumentacja Projektu**

**Spis treści**

[Wprowadzenie 4](#_Toc168137406)

[Cel Projektu 4](#_Toc168137407)

[Zakres Projektu 4](#_Toc168137408)

[Architektura Systemu 4](#_Toc168137409)

[Diagramy 4](#_Toc168137410)

[Opis Komponentów 4](#_Toc168137411)

[Technologie i Narzędzia 4](#_Toc168137412)

[Funkcjonalności Systemu 4](#_Toc168137413)

[Opis Szczegółowy 4](#_Toc168137414)

[Diagramy Przypadków Użycia 4](#_Toc168137415)

[Wymagania Systemowe 4](#_Toc168137416)

[Wymagania Sprzętowe 4](#_Toc168137417)

[Wymagania Programowe 4](#_Toc168137418)

[Harmonogram Projektu 4](#_Toc168137419)

[Zespół Projektowy 4](#_Toc168137420)

[Procedury Testowania 4](#_Toc168137421)

[Plan Testów 5](#_Toc168137422)

[Rodzaje Testów 5](#_Toc168137423)

[Zarządzanie Projektem 5](#_Toc168137424)

[Zarządzanie Ryzykiem 5](#_Toc168137425)

[Załączniki 5](#_Toc168137426)

# **Wprowadzenie**

# **Cel Projektu**

# 

# **Zakres Projektu**

# 

# **Architektura Systemu**

# 

# **Diagramy**

# 

# **Opis Komponentów**

# 

# **Technologie i Narzędzia**

# 

# **Funkcjonalności Systemu**

* **main.py**
  + Importowane Moduły
    - time: Umożliwia pracę z czasem (np. opóźnienia).
    - class\_Deck: Moduł zawierający klasę Deck do obsługi talii kart.
    - F\_bot: Moduł zawierający funkcję obsługującą bota
    - random: Umożliwia generowanie liczb losowych.
    - players: Moduł zawierający klasę players do obsługi graczy.
    - os: Umożliwia np. czyszczenie ekranu konsoli
  + Zmienne Globalne
    - sum\_total: Pomaga w przechowywaniu aktualnej wartości pieniędzy, którą postawił gracz.
  + Funkcje
    - logo():
      * Tworzy logo gry w konsoli
    - get\_deck(n=0):
      * Tworzy listę talii kart.
      * Dla każdej talii w liczbie n:
      * Tworzy obiekt Deck z modułu class\_Deck.
      * Tasuje talię (p.Shuffle()).
      * Dodaje talię do listy tab\_deck.
      * Zwraca listę talii tab\_deck.
    - get\_card(tab, n):
      * Losowo dobiera kartę z jednej z talii w liście tab.
      * Jeśli talia jest pusta, usuwa ją z listy i zmniejsza licznik talii n.
      * Zwraca dobraną kartę.
    - get\_players(i, p, p\_total):
      * Tworzy listę graczy tab\_players.
      * Dla każdego gracza w liczbie i:
      * Tworzy obiekt players z początkową sumą pieniędzy p\_total i wkładem p.
      * Dodaje gracza do listy tab\_players.
      * Funkcja zwraca listę graczy.
    - split(tab\_players, number\_player, total):
      * Wykonuje akcję "split" na wskazanym graczu number\_player.
    - player\_pass(tab\_players, number\_player):
      * Gracz pasuje, traci pieniądze w wysokości swojego zakładu, i jest usuwany z listy graczy.
    - win(tab\_win):
      * Dodaje wartość wygranej sum\_total dla graczy w liście tab\_win.
  + Główna Funkcja if \_\_name\_\_ == "\_\_main\_\_":
    - Obsługa głównej pętli gry:
      * Na samym początku są sprawdzane wszystkie możliwości wygrania i przegrania gracza.
      * Następnie losowane są karty dla Krupiera, bota i gracza
      * Na 3 turze: bot i gracz mają możliwość: podbicia stawki lub wyciągnięcia 3 karty
      * Po zakończonej grze, gracz może pozostać w grze i zacząć nową rozgrywkę z aktualną wartością pieniędzy.
* **Class\_Card.py**
  + Klasa Card
    - Stałe
      * Konstruktor \_\_init\_\_
    - def \_\_init\_\_(self, window, rank, suit, value):
      * window: Okno Pygame, w którym karta będzie wyświetlana.
      * rank: Ranga karty (np. "As", "Król", "2").
      * suit: Kolor karty (np. "Pik", "Karo").
      * value: Wartość karty (liczba całkowita).
    - cardName: Nazwa karty, złożona z jej rangi i koloru.
  + Metody
    - getName(self): Zwraca nazwę karty (rank + suit).
    - getValue(self): Zwraca wartość karty.
    - getSuit(self): Zwraca kolor karty.
    - getRank(self): Zwraca rangę karty.
* **Class\_Deck.py**
* Importowane Moduły
  + random: Umożliwia generowanie liczb losowych, w tym przypadku używane do tasowania talii.
  + class\_Card: Moduł zawierający klasę Card, która reprezentuje pojedynczą kartę w talii.
* Stałe
  + color: Krotka zawierająca kolory kart ("karo", "pik", "karo", "trefl").
  + BLACKJACKDictionary: Słownik zawierający rangi kart i ich wartości punktowe zgodnie z zasadami Blackjacka.
* Konstruktor \_\_init\_\_
  + def \_\_init\_\_(self, window, ValueDictionary=BLACKJACKDictionary):
    - window: Okno Pygame, w którym karty będą wyświetlane (Przygotowane pod rozbudowę gry).
    - ValueDictionary: Słownik zawierający rangi kart i ich wartości punktowe. Domyślnie jest to BLACKJACKDictionary.
    - self.startDeck: Lista zawierająca wszystkie karty w talii.
    - self.playingDeck: Lista zawierająca karty aktualnie używane w grze.
      * Tworzy pełną talię kart, iterując przez kolory (Deck.color) i rangi kart z ValueDictionary. Każda karta jest tworzona jako obiekt klasy Card z modułu class\_Card.
* Metody
  + Shuffle(self): Tasuje talię kart.
    - Kopiuje self.startDeck do self.playingDeck.
    - Ustawia wszystkie karty na stan "zakryte" (card.conceal())(Przygotowane pod rozbudowę gry).
    - Tasuje karty w self.playingDeck używając random.shuffle().
  + getCard(self): Dobiera kartę z talii.
    - Jeśli self.playingDeck jest pusta, zwraca False.
    - Usuwa i zwraca ostatnią kartę z self.playingDeck.
  + returnCard(self, card): Zwraca kartę do talii.
    - Wstawia kartę na początek self.playingDeck.
* **Game\_screen.py** (Przygotowane pod rozbudowę gry)
* Importowane Moduły
  + pygame: Biblioteka Pygame do tworzenia gier.
  + sys: Moduł systemowy Pythona, używany do zakończenia programu.
* Funkcja main
  + Argumenty
    - screen: Obiekt pygame.Surface, który reprezentuje główny ekran gry.
  + Opis działania:
    - def main(screen):
      * Przechwytywanie Zdarzeń:
        + for event in pygame.event.get():

Pętla iterująca przez wszystkie zdarzenia w kolejce zdarzeń Pygame.

* + - * Obsługa Zamykania Okna:
        + if event.type == pygame.locals.QUIT: pygame.quit() sys.exit()

Jeśli typ zdarzenia to QUIT (kliknięcie przycisku zamknięcia okna), Pygame jest zamykany (pygame.quit()), a program kończy działanie (sys.exit()).

* + - * Kod Gry:
        + print("gra")

W obecnej formie funkcja po prostu drukuje "gra" przy każdym zdarzeniu. W rzeczywistej grze tutaj znajdowałby się kod obsługujący logikę gry.

* + - * Zwracanie Stanu:
        + return 1

Funkcja zwraca wartość 1, co może być używane do określenia stanu gry lub następnego ekranu, który ma być wyświetlany.

* **menu\_screen.py** (Przygotowane pod rozbudowę gry)
* Importowane Moduły
  + pygame: Biblioteka Pygame do tworzenia gier.
  + sys: Moduł systemowy Pythona, używany do zakończenia programu.
* Funkcja main
  + Argumenty
    - screen: Obiekt pygame.Surface, który reprezentuje główny ekran gry.
  + Opis Działania
    - def main(screen):
      * Przechwytywanie Zdarzeń:
        + for event in pygame.event.get():

Pętla iterująca przez wszystkie zdarzenia w kolejce zdarzeń Pygame.

* + - * Obsługa Zamykania Okna:
        + if event.type == pygame.locals.QUIT: pygame.quit() sys.exit()

Jeśli typ zdarzenia to QUIT (kliknięcie przycisku zamknięcia okna), Pygame jest zamykany (pygame.quit()), a program kończy działanie (sys.exit()).

* + - * Obsługa Kliknięć Myszy:
        + if event.type == pygame.locals.MOUSEBUTTONDOWN and event.button == 1:

Sprawdza, czy zdarzenie to naciśnięcie lewego przycisku myszy (event.button == 1).

* + - * Sprawdzanie Obszarów Kliknięcia:
        + Obszar 1:

if 705 < event.pos[0] < 1214 and 341 < event.pos[1] < 1033: return 1

Sprawdza, czy kursor myszy znajduje się w prostokącie z współrzędnymi od (705, 341) do (1214, 1033). Jeśli tak, funkcja zwraca 1.

* + - * + Obszar 2:

elif 1394 < event.pos[0] < 1814 and 500 < event.pos[1] < 1033: return 2

Sprawdza, czy kursor myszy znajduje się w prostokącie z współrzędnymi od (1394, 500) do (1814, 1033). Jeśli tak, funkcja zwraca 2.

* + - * + Obszar 3:

elif 103 < event.pos[0] < 519 and 500 < event.pos[1] < 1033: return 3

Sprawdza, czy kursor myszy znajduje się w prostokącie z współrzędnymi od (103, 500) do (519, 1033). Jeśli tak, funkcja zwraca 3.

* + - * Domyślne Zwracanie:
        + return 0

Jeśli żaden z powyższych warunków nie jest spełniony, funkcja zwraca 0.

* **player.py**
* Definicja Klasy:
  + Klasa Player jest definiowana za pomocą słowa kluczowego class.
* Metoda \_\_init\_\_:
  + Konstruktor \_\_init\_\_ inicjuje nowego gracza.
    - Przyjmuje argumenty total\_player i prices, z których total\_player oznacza ilość pieniędzy, a prices reprezentuje aktualny zakład gracza.
    - Inicjalizuje atrybuty total\_player, prices, cards (lista kart gracza) i sum\_value (suma wartości kart gracza).
* Metoda split:
  + Metoda split umożliwia graczowi podzielenie się zakładem.
  + Przyjmuje argument total, który określa kwotę, jaką gracz chce podzielić się ze stołem.
  + Jeśli gracz ma wystarczającą ilość pieniędzy (total\_player >= total), dodaje kwotę do zakładu (prices) i odejmuje ją od całkowitej ilości pieniędzy gracza (total\_player), zwracając True.
  + W przeciwnym razie zwraca False.
* Metoda get\_card:
  + Metoda get\_card dodaje kartę do ręki gracza.
  + Przyjmuje argumenty card (obiekt karty) i value (wartość karty).
  + Dodaje wartość karty do sumy wartości kart gracza (sum\_value) oraz kartę do listy kart gracza (cards).
* Metoda win:
  + Metoda win aktualizuje stan gracza po wygranej rundzie.
  + Przyjmuje argument total, który określa wygraną sumę pieniędzy.
  + Dodaje wygraną sumę do całkowitej ilości pieniędzy gracza (total\_player), zeruje zakład (prices), czyści listę kart gracza (cards) oraz resetuje sumę wartości kart gracza (sum\_value).
* Metoda get\_prices i get\_sum\_value:
  + Metoda get\_prices zwraca aktualny zakład gracza.
* Metoda get\_sum\_value zwraca sumę wartości kart gracza.
* **settings\_screen.py** (Przygotowane pod rozbudowę gry)
* Przechwytywanie Zdarzeń:
  + Pętla for iteruje przez wszystkie zdarzenia w kolejce zdarzeń Pygame, które zostały zgromadzone od ostatniego wywołania funkcji pygame.event.get().
  + Każde zdarzenie jest analizowane w pętli.
* Obsługa Zamknięcia Okna:
  + Warunek if event.type == pygame.locals.QUIT: sprawdza, czy zdarzenie to zamknięcie okna (czy użytkownik kliknął przycisk zamknięcia okna).
  + Jeśli tak, wywoływane są funkcje pygame.quit() i sys.exit(), które odpowiednio zamykają bibliotekę Pygame i kończą działanie programu.
* Kod Ustawień:
  + Wewnątrz pętli for znajduje się kod związany z obsługą ustawień gry.
  + W tym przypadku kod został zastąpiony przez prostą instrukcję print("ustawienia"), która wypisuje "ustawienia" w konsoli.
* Zwracanie Stanu:
  + Na koniec funkcja zwraca wartość 2.
* **f\_bot.py**
* Funkcja f\_bot:
  + Przyjmuje dwa argumenty: sum (całkowita wartość kart bota) i card (aktualna karta Krupiera).
  + Po przeanalizowaniu tych wartości funkcja podejmuje decyzje o działaniu
  + Funkcja zwraca wartości 1, 2 lub 3, oznaczające kolejno: Hit, Stand i Double down
* **tutorial\_screen.py** (Przygotowane pod rozbudowę gry)
* Przechwytywanie Zdarzeń:
  + Pętla for iteruje przez wszystkie zdarzenia w kolejce zdarzeń Pygame, które zostały zgromadzone od ostatniego wywołania funkcji pygame.event.get().
  + Każde zdarzenie jest analizowane w pętli.
* Obsługa Zamknięcia Okna:
  + Warunek if event.type == pygame.locals.QUIT: sprawdza, czy zdarzenie to zamknięcie okna (czy użytkownik kliknął przycisk zamknięcia okna).
  + Jeśli tak, wywoływane są funkcje pygame.quit() i sys.exit(), które odpowiednio zamykają bibliotekę Pygame i kończą działanie programu.
* Kod Samouczka:
  + Wewnątrz pętli for znajduje się kod związany z obsługą samouczka gry.
  + W tym przypadku kod został zastąpiony przez prostą instrukcję print("samouczek"), która wypisuje "samouczek" w konsoli.
* Zwracanie Stanu:
  + Na koniec funkcja zwraca wartość 3.
  + Może to oznaczać, że po wywołaniu tej funkcji gracz powinien być przekierowany do ekranu samouczka.

# **Opis Szczegółowy**

# 

# **Diagramy Przypadków Użycia**

# 

# **Wymagania Systemowe**

# 

# **Wymagania Sprzętowe**

# 

# **Wymagania Programowe**

# 

# **Harmonogram Projektu**

# 

# **Zespół Projektowy**

# 

# **Procedury Testowania**

# 

# **Plan Testów**

# 

# **Rodzaje Testów**

# 

# **Zarządzanie Projektem**

# 

# **Zarządzanie Ryzykiem**

# 

# **Załączniki**