

## ***1. Cel i opis projektu***

Celem projektu było stworzenie gry „Gobblet Gobblers” – rozszerzonej wersji gry kółko i krzyżyk w której gracz ma do dyspozycji po kilka pionków, które mogą przykrywać się wzajemnie. Podczas ruchu gracz może wybrać pionek z puli dostępnych pionków lub przenieść swój pionek znajdujący się na planszy na inne pole, pod warunkiem, że nie znajduje się tam jego własny pionek lub większy pionek przeciwnika.

## ***2. Podział programu na klasy i opis klas***

**Klasy w programie można podzielić na dwie grupy:**

Wyjątki własne:

- OutOfPawnsError (Exception) – sygnalizacja pustej listy dostępnych pionków
- WrongLengthError (Exception) – podana długość planszy jest z poza przedziału od 3 do 26
- CannotMovePawnError (Exception) – sytuacja, gdy komputer nie może wykonać ruchu w zależności od okoliczności
- WrongFieldError (KeyError) – podane przez użytkownika pole nie istnieje
- HigherOrTheSameRankError (Exception) – próba przykrycia pionka przeciwnika swoim własnym pionkiem o niższej randze
- CoveringYourOwnPawnError (Exception) – próba przykrycia własnego pionka
- TakingOpponentPawnError (Exception) – próba przemieszczenia pionka przeciwnika
- ChoosingEmptyFieldError (Exception) – wybranie pustego pola jako pola, z którego chce się zabrać pionek

Klasy własne zdefiniowane:

- GenerateField () – odpowiada za przygotowanie planszy do gry na podstawie podanej długości żądanego pola o kształcie kwadratu
- Player () – tworzy obiekt gracza o danym symbol i danej liczbie pionków
- Computer (Player) – tworzy obiekt komputera jak jeden z graczy o takich samych parametrach jak gracz, ale z nieco zmienionym sposobem wykonywania ruchów

## ***3. Instrukcja użytkownika i opis plików***


**Opis plików:**

Program składa się z następujących plików:

- Project\_Exceptions.py – zawiera zdefiniowane wyjątki
- Project\_Classes.py – zawiera zdefiniowane klasy
- Project\_DisplayField.py – zawiera funkcję wyświetlającą planszę
- Project\_CheckWinner.py – zawiera funkcje sprawdzające czy wyłoniono zwycięzcę
- Project\_GamemodeRounds.py – zawiera funkcje obsługujące poszczególne rundy (pierwszą i dalsze) w danym trybie
- Project\_HelpingGamemodesFunction.py – zawiera funkcje stanowiące części wspólne dla funkcji ‘pve’ i ‘pvp’ znajdujących się w pliku poniżej
- Project\_GamemodeFunctions.py – zawiera funkcje odpowiadające za dany tryb gry
- Project\_ActualGame.py – zawiera główną funkcję odpowiadającą za start i przebieg gry
- test\_Project\_Exceptions.py – zawiera testy do sytuacji wyjątkowych
- test\_Project\_Classes.py – zawiera testy do prawidłowych ruchów gracza i komputera
- test\_Project\_CheckWinner.py – zawiera testy do funkcji sprawdzających zwycięzcę

\* wszystkie pliki testowe zawierają przeprowadzenie symulacji wszystkich możliwych sytuacji jakie możemy napotkać w grze na poziomie logiki programu

### Instrukcja użytkownika:

Aby rozpocząć grę należy uruchomić plik 'Project\_ActualGame.py'. Po uruchomieniu należy nacisnąć przycisk  w prawym górnym rogu. Po naciśnięciu przycisku wyświetli się zapytanie o wybranie trybu gry („Gracz vs Gracz” lub „Gracz vs Komputer”) lub możliwość wyjścia z gry w zależności od znaku wpisanego do konsoli przez użytkownika. Znaki te są wyświetlane w nawiasach kwadratowych:

```
(Player vs Player)[1] or (Player vs Computer)[2] or Leave[Any Key]:
```

Po wybraniu trybu gry użytkownik zostanie zapytany o żądany rozmiar tabeli:

```
How large game board shall be:
```

Dostępne długości boku kwadratowej planszy to przedział od 3 do 26 włącznie. Gdy użytkownik nie wpisze liczby, pojawi się komunikat:

```
How large game board shall be:dsg\
Wrong input!
How large game board shall be:
```

„Wrong input!” i ponownie zapyta użytkownika

o rozmiar. Jeżeli zostanie wpisana liczba poniżej 3 to wyświetli się komunikat:

```
Field's length cannot be negative,zero,one or two!
```

a jeżeli wpisze się liczbę

powyżej 26 to zostanie wyświetlony komunikat:

```
How large game board shall be:28
```

```
Max available size is 26!
```

Gdy użytkownik wprowadzi już rozmiar tablicy

to gracz lub gracz o numerze 1 (w zależności od trybu gry) zostanie zapytany o wybór symbolu „x” lub „o”:

```
Player 1, choose symbol: (x/o):
```

```
sdf
```

```
This symbol is not available!
```

Po poprawnym wprowadzeniu wszystkich

parametrów rozgrywki w konsoli pojawi się plansza i listy dostępnych pionków dla obu graczy:

```

  A      B      C
-----
1 |      |      |      |
-----
2 |      |      |      |
-----
3 |      |      |      |
-----

Player pawns:['1x', '1x', '2x', '2x', '3x', '3x']
Computer pawns:['1o', '1o', '2o', '2o', '3o', '3o']
Player, choose field:
```

lub

```

      A      B      C
-----
1 |   |   |   |
-----
2 |   |   |   |
-----
3 |   |   |   |
-----

Player 1 pawns:['1x', '1x', '2x', '2x', '3x', '3x']
Player 2 pawns:['1o', '1o', '2o', '2o', '3o', '3o']
Player 1, choose field:

```

w zależności od trybu gry i rozmiaru planszy. Liczba pionków dla każdego gracza wynosi 2 x rozmiar planszy (n), po dwa pionki o każdej randze od 1 do n.

W pierwszej rundzie gracze mogą tylko wybrać pionek z listy i wstawić go na odpowiednie pole podając współrzędne (tak jak w grze w statki). Przykład:

```

Player 2, choose field:A3
Player 2, choose pawn:1o

```

Po zatwierdzeniu klawiszem „ENTER” konsola wyświetli aktualny stan gry (sytuację na planszy oraz liczbę pionków dla każdego z graczy):

```

      A      B      C
-----
1 | 1x |   |   |
-----
2 |   |   |   |
-----
3 | 1o |   |   |
-----

Player 1 pawns:['1x', '2x', '2x', '3x', '3x']
Player 2 pawns:['1o', '2o', '2o', '3o', '3o']

```

W następnych rundach gracze mogą już wybrać typ ruchu (wybrać pionek z listy lub przestawić swój pionek na inne pole na planszy):

```

Player 1, put pawn from list[L] or move one from field[F]:

```

wybierając odpowiednią opcję podaną w nawiasach kwadratowych. Gdy gracz wybierze opcję „L” lub „l” to wykonuje się ruch taki sam jak w pierwszej rundzie. Natomiast jeżeli wybierze się opcję „F” lub „f” to gra zapyta najpierw o to z jakiego pola gracz chce ruszyć pionek, a potem o pole, na które chce postawić pionek:

```

Player 1, put pawn from list[L] or move one from field[F]: f
Player 1, choose field where you want to move pawn from: A1
Player 1, choose field where you want to move pawn to: B2

```

. Można również przykryć pionek przeciwnika:

```

      A      B      C
-----
1 |   |   |   |
-----
2 |   | 1x |   |
-----
3 | 1o |   |   |
-----

Player 1 pawns:['1x', '2x', '2x', '3x', '3x']
Player 2 pawns:['1o', '2o', '2o', '3o', '3o']
Player 2, put pawn from list[L] or move one from field[F]: 1
Player 2, choose field:B2
Player 2, choose pawn:3o

```

, gdzie po

zatwierdzeniu sytuacji na planszy będzie wyglądać tak:

```

      A      B      C
-----
1 |   |   |   |
-----
2 |   | 3o |   |
-----
3 | 1o |   |   |
-----

Player 1 pawns:['1x', '2x', '2x', '3x', '3x']
Player 2 pawns:['1o', '2o', '2o', '3o']
Player 1, put pawn from list[L] or move one from field[F]:

```

W ten sposób gra toczy się do momentu, gdy któryś z graczy nie wypełni danego wiersza, danej kolumny, lub danej przekątnej tylko swoimi pionkami. Po wyłonieniu zwycięzcy gra wyświetli komunikat wyświetlający zwycięski symbol np.:

The winning symbol is: o.

### Sytuacje wyjątkowe podczas rozgrywki:

Zdarza się również, że użytkownik pomyli się przy wyborze danej opcji. W takich sytuacjach w projekcie pojawią się odpowiednie komunikaty oraz zainicjowane zostaną odpowiednie działania:

```

Player 1, choose field:f
Wrong coordinate!
Player 1, choose field:

```

, gdy użytkownik wprowadzi nieistniejące pole

```

Player 1, choose pawn:2qt
You do not have that pawn as available one!
Player 1, choose pawn:

```

, gdy użytkownik poda nie swój lub nieistniejący pionek

```

Player 1, put pawn from list[L] or move one from field[F]: sdf
Wrong input!
Player 1, put pawn from list[L] or move one from field[F]:

```

, gdy gracz wybierze nieistniejącą opcję

	A	B	C
1	1x		
2			
3			3o

Player 1 pawns:['1x', '2x', '2x', '3x', '3x']  
 Player 2 pawns:['1o', '1o', '2o', '2o', '3o']  
 Player 1, put pawn from list[L] or move one from field[F]: sdf  
 Wrong input!  
 Player 1, put pawn from list[L] or move one from field[F]: 1  
 Player 1, choose field:C3  
 Player 1, choose pawn:2x  
 You cannot cover pawn with the same or higher rank!  
 Player 1, choose field:

, gdy gracz  
 będzie chciał przykryć pionek przeciwnika o wyższej lub tej samej randze

	A	B	C
1	1x		
2			
3			3o

Player 1 pawns:['1x', '2x', '2x', '3x', '3x']  
 Player 2 pawns:['1o', '1o', '2o', '2o', '3o']  
 Player 1, put pawn from list[L] or move one from field[F]: sdf  
 Wrong input!  
 Player 1, put pawn from list[L] or move one from field[F]: 1  
 Player 1, choose field:C3  
 Player 1, choose pawn:2x  
 You cannot cover pawn with the same or higher rank!  
 Player 1, choose field:A1  
 Player 1, choose pawn:3x  
 Cannot cover your own pawn!  
 Player 1, choose field:

, gdy gracz  
 będzie chciał przykryć swój pionek

	A	B	C
1	1x		
2		3x	
3			3o

Player 1 pawns: ['1x', '2x', '2x', '3x']  
 Player 2 pawns: ['1o', '1o', '2o', '2o', '3o']  
 Player 2, put pawn from list[L] or move one from field[F]: f  
 Player 2, choose field where you want to move pawn from: sdf  
 Player 2, choose field where you want to move pawn to: sdf  
 Wrong field!  
 Player 2, choose field where you want to move pawn from: f  
 Player 2, choose field where you want to move pawn to: A3  
 Wrong field!  
 Player 2, choose field where you want to move pawn from: C3  
 Player 2, choose field where you want to move pawn to: sd  
 Wrong field!  
 Player 2, choose field where you want to move pawn from: █

, gdy któreś z pól podczas ruchu z pola na pole nie istnieje

Player 2, choose field where you want to move pawn from: C2  
 Player 2, choose field where you want to move pawn to: A1  
 First chosen field is empty!  
 Player 2, choose field where you want to move pawn from: █

, gdy pierwsze wybrane pole jest puste

Player 2, choose field where you want to move pawn from: B2  
 Player 2, choose field where you want to move pawn to: A1  
 You cannot take opponent pawn!  
 Player 2, choose field where you want to move pawn from: █

, gdy pierwsze wybrane pole zawiera pionek przeciwnika

Player 1, choose field where you want to move pawn from: A1  
 Player 1, choose field where you want to move pawn to: B2  
 You cannot cover pawn with the same or higher rank!

, gdy gracz próbuje przykryć pionek przeciwnika takiej samej lub wyższej randze

```

      A      B      C
-----
1 | 1x |   |   |
-----
2 | 3o |   | 2o |
-----
3 |   |   | 3x |
-----

Player pawns:['1x', '2x', '2x', '3x']
Computer pawns:['1o', '1o', '2o', '3o']
Player, put pawn from list[L] or move one from field[F]:f
Player, choose field where you want to move pawn from: C3
Player, choose field where you want to move pawn to: A1
Cannot cover your own pawn!
Player, choose field where you want to move pawn from: A1
Player, choose field where you want to move pawn to: C3
Cannot cover your own pawn!
Player, choose field where you want to move pawn from: █

```

,gdy gracz

próbuję przykryć swój pionek

Ponadto, gdy lista pionków jest już pusta:

```

Player pawns:[]
Computer pawns:['2o']
Player, put pawn from list[L] or move one from field[F]:█

```

a gracz wybierze opcję wyboru pionka z listy to gra automatycznie przełączy się na opcję ruchu z pola na pole:

```

Player, put pawn from list[L] or move one from field[F]:1
Player, choose field where you want to move pawn from: A1
Player, choose field where you want to move pawn to: C1█

```

W przypadku gry z komputerem, komputer sam jest w stanie skorygować swoje działanie w sytuacji wyjątkowej. Zasada działania komputera opiera się na losowym wybieraniu parametrów potrzebnych do wykonania ruchu.

#### 4. Podsumowanie:

Zakres prac nad projektem obejmował stworzenie: odpowiednich klas oraz uwzględnienie wszystkich przypadków i sytuacji wyjątkowych ich dotyczących, funkcji sprawdzających zwycięzcę, funkcji wyświetlającej planszę, funkcji obsługujących dany tryb gry i związanych z nimi funkcji pomocniczych, funkcji działającej jako interfejs (pobierającej odpowiednie parametry gry i uruchomienie wybranego trybu gry).

Jedyną rzeczą jakiej nie udało się osiągnąć jest wydzielenie pętli obsługujących rundy w funkcjach odpowiadających za działanie odpowiedniego trybu. Mianowicie wydzielenie tych pętli w znaczącym stopniu zaburza działanie gry.

Nieprzewidzianą przeszkodą było testowanie metod klasy komputer, gdyż są one oparte na mechanizmach losujących, jednakże po pewnej ilości prób napisanie odpowiednich testów i zrobienie ich deterministycznymi zakończyło się sukcesem. Projekt został zrealizowany zgodnie z przyjętymi na początku założeniami.

## **Zalety projektu:**

Projekt gry „Gobblet Gobblers” ma wiele zalet. Najważniejszą z nich jest intuicyjny interfejs użytkownika, który w razie ewentualnego błędnego wprowadzenia danych wyświetla odpowiedni komunikat, mówiący jaką niedozwoloną akcją próbuje wykonać gracz. Wyświetlana plansza jest bardzo przejrzysta i łatwa, jeżeli chodzi o lokalizację odpowiednich pól. Co więcej, jest w pełni konfigurowalna, a dla graczy chcących prawdziwego wyzwania, można ustawić rozmiar planszy dochodzący nawet do 26 jednostek. Dzięki możliwości wyboru trybu gry, gra jest wspaniałą opcją dla pojedynczego gracza lub dwóch. Dzięki dobrze opisanej instrukcji użytkownik szybko pozna zasady rządzące grą.

Sama gra zaprogramowana jest z użyciem prostych mechanizmów programowania dzięki czemu początkowy programista nie miałby problemu z interpretacją kodu. Kod jest opatrzony również przydatnymi komentarzami w razie wątpliwości co do zasad działania programu. Sama gra jest również podzielona na odpowiednie pliki o danych kategoriach, dzięki czemu poruszanie się po kodzie nie stanowi żadnego kłopotu.