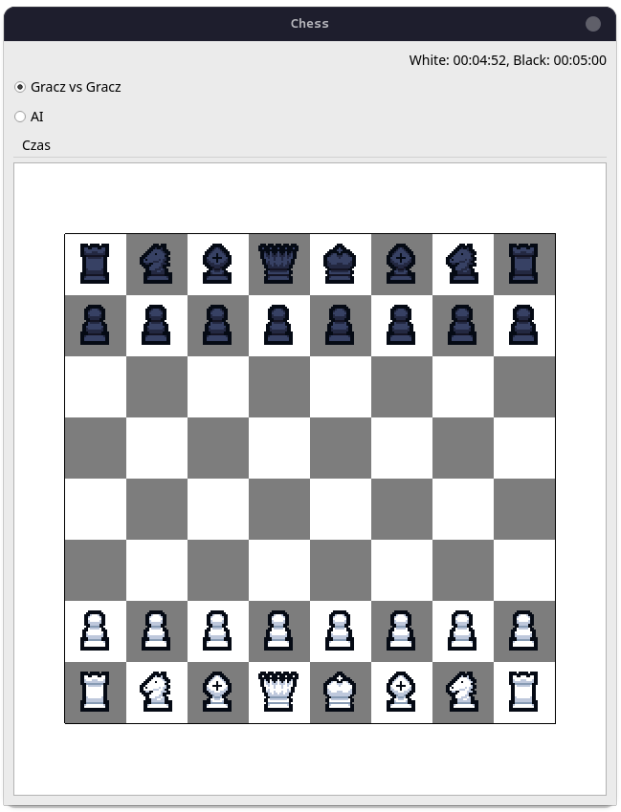
Laboratorium Architektura Systemów Komputerowych - Sprawozdanie 6	
Mateusz Drzewiecki, 185607 Mikołaj Galant, 188565	Zadanie 6 - Szachy z przeciwnikiem AI

Wprowadzenie

Projekt umożliwia grę w szachy dla dwóch osób lokalnie bądź grania na przeciwnika, który posługuje się algorytmem Q-Learning w celu podejmowania decyzji o ruchu. Możliwe jest ustawienie zegara szachowego na kilka popularnych trybów rozgrywki. Projekt został napisany w języku Python z wykorzystaniem bibliotek Qt.

Rozgrywka i elementy interfejsu

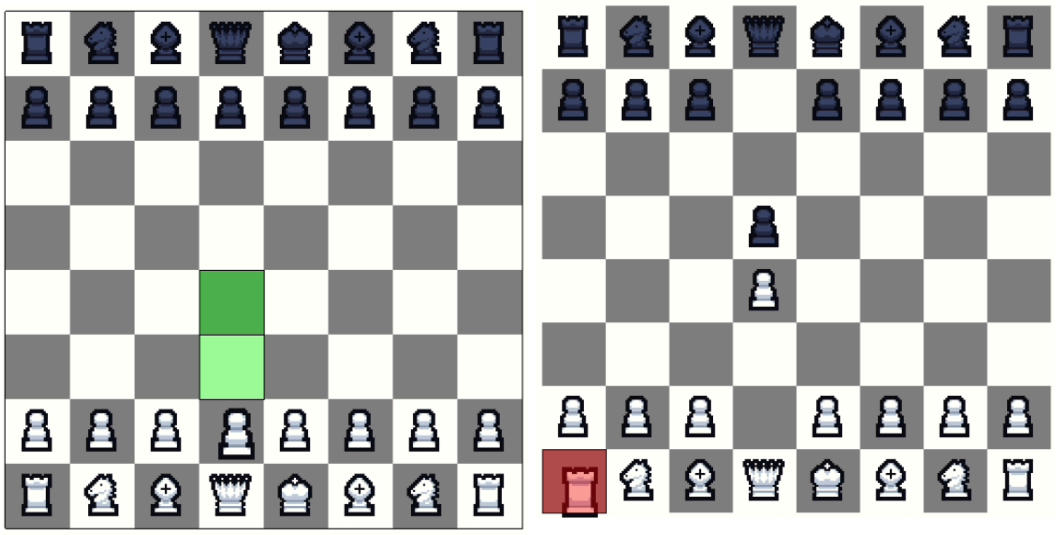
Zasad związanych z szachami nie trzeba przedstawiać. W naszej wersji gry nie zaimplementowano promocji pionów, roszad ani bicia w przelocie. Gra rozpoczyna się od rozstawionych figur na wyjściowych pozycjach.



Rys. 1. Interfejs gry po rozpoczęciu rozgrywki

Ruch dokonuje się poprzez przeciągniecie figury myszką. Chwycenie figury jest sygnalizowane poprzez niewielkie powiększenie jej. Program zaznacza wszystkie możliwe

ruchy do wyboru kolorem zielonym. Gdy brak dostępnych ruchów zaznacza pole okupowane przez figurę kolorem czerwonym.

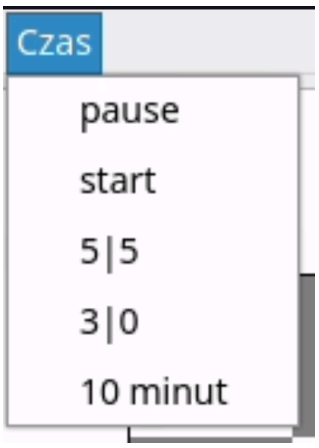


Rys. 2. Efekty chwycenia figur z rysowaniem możliwych ruchów.

Gra toczy się do aż do zbicia jednego z królów, co zwraca komunikat o zakończeniu gry.

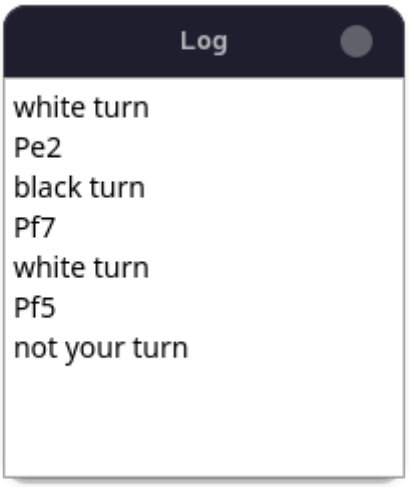
Rys. 3. Okno końca gry.

Nad planszą widnieje pasek menu podpisany “Czas”. Z jego poziomu można wybrać tryb gry i wpływać na zegar znajdujący się w prawym górnym rogu okna. Zapis “5 | 5” oznacza, że bazowo każdy z graczy ma po bazowo 5 minut na rozgrywkę, a po wykonaniu ruchu do czasu doliczane jest 5 sekund. Analogicznie, “3 | 0” oznacza, że każdy z graczy ma 3 minuty, a po ruchu nie jest dodawany czas. Pozostałe opcje do wyboru to “start”, “pause” oraz “10 minut”.



Rys. 4. Menu Czas

Oknu gry towarzyszy również okno komunikatów. Na nim wyświetlane są informacje o grze takie jak którego gracza jest tura albo jaki ruch został wykonany.



Rys. 5. Okno z komunikatami gry

W lewym górnym rogu znajdują się radiobuttony. Dzięki nim można przełączyć grę na rozgrywkę przeciwko komputerowi. Steruje on zawsze czarnymi figurami.

Opis kodu i przeciwnika AI

Program składa się z kilku klas odpowiadających m.in. za wyświetlanie interfejsu, logikę ruchów, piony, zegar, AI. Jednym z większych wyzwań było napisanie funkcji sprawdzającej dopuszczalne danej figury z danej pozycji. Zwraca ona listę dozwolonych ruchów na podstawie nazwy figury, jej pozycji i jej otoczenia. Przykład dla pionów został przedstawiony na poniższym obrazku

```python
def getValidMoves(self, write):
    valid_moves = []
    current_row, current_col = self.currentPosition()
    direction = -1 if self.color == "white" else 1
    items = self.scene().items()

    if self.piece_type == "pawn":

        if 0 <= current_row + direction <= 7:
            valid_moves.append((current_row + direction, current_col))

            # pierwszy ruch piona
            if (current_row == 1 and self.color == "black") or (current_row == 6 and self.color == "white"):
                valid_moves.append((current_row + 2 * direction, current_col))

        #sprawdź czy na przekątnych polach są figury przeciwnika, jeżeli tak to dodaj do możliwych ruchów
        for col_offset in [-1, 1]:
            target_row = current_row + direction
            target_col = current_col + col_offset
            if 0 <= target_row <= 7 and 0 <= target_col <= 7:
                for item in items:
                    if isinstance(item, ChessPiece):
                        items_row = int(item.y() / 64)
                        items_col = int(item.x() / 64)
                        if item.color != self.color and items_row == target_row and items_col == target_col:
                            valid_moves.append((items_row, items_col))

        #sprawdź czy na przeciwnym polu jest figura
        for item in items:
            if isinstance(item, ChessPiece):
                items_row = int(item.y() / 64)
                items_col = int(item.x() / 64)
                if item.color != self.color:
                    if items_row == current_row + direction and items_col == current_col:
                        valid_moves.remove((current_row + direction, current_col))
                        break
```

Rys. 6. Wyznaczanie dozwolonych ruchów na przykładzie piona.

Do każdej figury podejście było indywidualne z uwagi na unikatowość zachowań. Dało to w efekcie funkcję na ok. 130 linijek, co stanowi niemal 11% objętości całego kodu. Niemniej, funkcja jest bardzo potrzebne i stanowi często integralną część działania innych funkcji np. sprawdzania czy nastąpi bicie figury po zajęciu jej miejsca.

```python
def isNotOccupied(self):
        target_items = self.scene().items(self.mapToScene(self.boundingRect().center()))
        for item in target_items:
            #sprawdzenie czy przeciwnik nie stoi na polu docelowym
            if isinstance(item, ChessPiece):
                if item.color != self.color:
                    self.scene().removeItem(item)
                    print("AI zbiło figurę") if self.color =="black" and chessboard.ai!=None else None
                    chessboard.addMessage("AI zbiło figurę") if self.color =="black" and chessboard.ai!=None else None
                    if chessboard.ai!=None:
                        chessboard.ai.epsilon = 0 if self.color =="black" else None
                    return True
            #sprawdzenie czy sojusznik nie stoi na polu docelowym
            elif isinstance(item, QGraphicsRectItem):
                self.setPos(self.previous_col * 64, self.previous_row * 64)
                return False
        else:
            return True
```

Rys. 7. Funkcja sprawdzające bicie figury.

Podobnie jak inne aspekty programu, AI przeciwnika zostało zrealizowane w postaci klasy posiadającej metody związane z algorytmem Q-learning. Po włączeniu jej, rozpoczyna ona tworzyć tablicę Q, składającą się ze stanu, akcji i wartości przypisanej tej pary. Stan został ujęty jako aktualna pozycja figury, a akcja jako możliwe kolejne położenie figury po wykonaniu ruchu. Z początku wartości tabeli Q są równe zero, ale wraz z postępem rozgrywki się zmieniają na podstawie równania Bellmana - wartość Q dla danej pary

stan-akcja jest równa sumie wartości natychmiastowej nagrody za wykonanie akcji oraz maksymalnej wartości Q dla nowego stanu. Za wykonanie ruchu przyznawana jest nagroda podstawowa opisana przez to równanie, a zbicie jest nagradzane dodatkowymi punktami zależnymi od rodzaju zbitej figury. Ma to na celu spowodować, by AI było bardziej skore do zbijania figur przeciwnika.
AI dokonuje ruchu na podstawie odmiany metody epsilon-gready. Za każdym razem losowana jest wartość liczbowa z zakresu 0-1. Jeżeli wylosuje się mniejsza wartość niż przyjęta wartość epsilon, AI dokonuje losowego ruchu. W innym wypadku wykonuje akcję, której przypisano największą wartość Q. Wartość epsilon w ciągu rozgrywki jest zmienna np. jeżeli możliwe jest zbicie króla lub królowej gracza to epsilon ustawia się na 0, a wartość Q akcji związanej z biciem zwiększana jest o 1000. Po wyborze akcji stan tablicy Q aktualizuje się.

```
def choose_action(self):
    while True:
        print(f'dokonuję wyboru Q, {self.epsilon}')
        if self.powtorz or random.uniform(0, 1) < self.epsilon:
            key = random.choice(list(self.Q.keys()))
            value = self.Q[key]
            if value == None:
                self.powtorz = True
                continue
            else:
                if key[1] in self.last_valid_moves:
                    pos1, pos2= key
                    self.powtorz =False
                    return pos1, pos2, value
        else:

            max_value=0
            for key, value in self.Q.items():
                if value > max_value:
                    max_value = value
            if max_value == 0:
                choice = random.choice(list(self.Q.keys()))
                pos1, pos2 = choice
                return pos1, pos2, 0
            else:
                print(1)
                for key, value in self.Q.items():
                    if value == max_value:
                        if key[1] in self.last_valid_moves:
                            self.epsilon=0.6 if self.epsilon < 0.01 else None
                            pos1, pos2= key
                            return pos1, pos2, value
```

Rys. 8 Metoda wyboru ruchu przez AI.

Komentarz do kodu i porównanie ze środowiskiem .NET

W przeciwieństwie do zaprezentowanego środowiska .NET na zajęciach laboratoryjnych, Python nie oferuje swobody w postaci nie przejmowania się wątkami. Elementy nie związane z planszą - zegar i okno komunikatów - musiały zostać celowo

napisane w innych wątkach niż plansza i figur. W roli programisty jest pamiętanie o tym, w przeciwieństwie do środowiska .NET. W nim większość czynności związanych np. z dodaniem timera jest zautomatyzowana. Niemniej, Python daje możliwość pisania bardziej kompaktowego kodu - prezentowany program zmieścił się w około 800 linijkach, gdzie pierwszy projekt prezentowany na tym przedmiocie miał ich około 400 mimo znacznie mniejszej funkcjonalności.
Z wad samej implementacji AI trzeba zaznaczyć, że czasami zacina się nie mogąc podjąć decyzji jaki ruch wybrać. Należy wtedy kliknąć kilkukrotnie na jakąkolwiek figurę szachową w celu wymuszenia rozpoczęcia podejmowania nowej decyzji. Pojawia się również błąd, który pozwala pionom dokonać bicia figury stojącej naprzeciwko nich.