Projekt PIPR – Warcaby angielskie

Maksym Bieńkowski

325147

1. Cel i opis projektu

Celem projektu było stworzenie programu do gry w warcaby angielskie z trzema trybami – Gracz vs Gracz, Gracz vs Bot i Bot vs Bot, z naciskiem na bota jak najlepszej jakości. Zdecydowałem się na implementację graficznego interfejsu użytkownika w bibliotece pygame, udało mi się spełnić wszystkie wymagania projektowe. Jeśli chodzi o wspomniane sprawdzanie poprawności ruchów, po kliknięciu pionka program sam pokazuje użytkownikowi jego jedyne możliwe ruchy. Zgodnie z zasadami gry podlinkowanymi w poleceniu, oto kilka założeń, jako, że okazało się, że każdy z moich kolegów znał trochę inną wersję warcabów:):

1. Król porusza się jak pionek, jednak w dowolnym kierunku – nie może skakać kilka pól naraz, jak szachowy goniec.
2. Tylko król może bić do tyłu – niektórzy ze wspomnianych kolegów byli w szoku, że bicie pionkami do tyłu nie jest dozwolone.
3. Bicie jest forsowne, wielokrotne bicie tym samym pionkiem/królem trzeba wykonać w tej samej rundzie. Gra zmieni rundę dopiero, gdy poprzedni użytkownik wykonał ruch i, jeśli było to bicie, nie ma więcej możliwych bić bijącym pionkiem. Jeśli jest możliwe więcej niż jedno bicie, gracz ma wolność wyboru.

Jeśli chodzi o algorytm grający w grę, zdecydowałem się na implementację algorytmu minimax z ulepszeniem o alpha-beta pruning. Znacząco przyspieszyło to średni czas ruchu dla bota, ale o tym więcej w części refleksyjnej.

2. Podział programu na pliki oraz klasy

1. Foldery images i fonts – zawierają pliki .png oraz .otf używane do wyświetlania pionków oraz tekstu w oknie gry. Pliki .png są wynikiem moich zmagań z gimpem, a wszystkie inne czy ich składowe (gwiazdka króla) są na odpowiedniej licencji i mam prawo używać ich prywatnie.
2. Folder checkers – zawiera cały kod gry
    3. constants.py – plik zawierający wszystkie stałe używane w grze, dla ułatwienia ich zmiany. Między innymi znajdziemy tam kody rgb kolorów używanych przez pygame'a, szerokości w pikselach odpowiednich elementów czy ilość czasu, jaką bot ma przeczekać przed wypluciem ruchu, aby rozgrywka na niższej głębii lub z losowym botem była płynna.

    4. gui.py – te elementy graficznego interfejsu użytkownika, które nie były zależne od wewnętrznej logiki gry, jak ekran menu, czy ekran game over, znalazły tu swoje miejsce.

    5. piece_move_board.py – największy plik ze wszystkich, zawiera trzy zależne od siebie cyklicznie klasy reprezentujące pionka, ruch i planszę do gry. Kod w tym pliku, a konkretnie w klasach pionka i szachownicy, odpowiedzialny jest za większość logiki gry – sprawdzanie możliwych ruchów, liczenie, czyja jest runda, ewaluacja pozycji podawana botowi, czy obsługa wykonanego ruchu pionkiem na poziomie

wewnętrznym. Wartą wspomnienia częścią logiki przy wykonywaniu ruchów jest odświeżany po każdym ruchu słownik mapujący kolory graczy na ich pionki, z których każdy powiązany jest z możliwymi w danym momencie do wykonania ruchami.

6. player.py – zawiera trzy klasy reprezentujące graczy. Pierwsza, player, to podstawa do dziedziczących po niej botów – RandomBot oraz MinimaxBot. RandomBot to pewna zaszłość pozostała po tym, gdy starałem się stworzyć jakkolwiek działającą implementację czegoś, co możnaby nazwać algorytmem grającym w grę, jednak pozostawiam go do Pani dyspozycji, jest jakimś urozmaiceniem i dobrze bawiłem się, oglądając jego zmagania z botem minimaxowym.

7. field.py – zgodnie z początkową koncepcją odnośnie separacji klas, której nie udało mi się utrzymać, zawiera pojedynczą klasę reprezentującą jedno z 64 pól na szachownicy.

8. game.py – plik zawierający klasę gry. Łączy ona gui z wewnętrzną reprezentacją partii, kieruje przebiegiem partii ze strony graficznej – obsługuje kliknięcia na pola i pionki, wyświetla szachownicę i ją aktualizuje, pokazuje możliwe ruchy.

9. main.py – plik zawierający główną funkcję, której wywołanie otwiera okienko z grą. Zawiera główną pygame'ową pętlę, inicjalizowane są w nim instancje gry zależnie od wybranego trybu oraz zadawane pytania do użytkownika o głębie i rodzaj bota.

10. Folder testowy - zawiera testy do każdej klasy logiki wewnętrznej programu

11. readme.md – plik z instrukcjami odnośnie obsługi programu

3. Część refleksyjna

Zaczynając od rozwiązań, czy założeń, które się końcowo nie sprawdziły:

1. Początkowo, zainspirowany słowami prof. Wysoty i zachęcony przez Panią ze względu na ogrom poradników do pygame'a, chciałem zaimplementować warcaby po swojemu, w qt, jednak coraz to kolejne niepowodzenia i sugestie kolegów sprawiły, że postanowiłem spróbować przekonać się do pygame'a i był to przełom w postępie projektu. Po jednym wieczorze spędzonym na czterogodzinnym poradniku, nie potrzebowałem zbytnio nic więcej, aby napisać całą funkcjonującą, graficzną stronę gry. Okazało się, że nie bez powodu każde warcaby i szachy w pythonie, które znalazłem przy researchu były napisane w pygame…
2. Początkowo, zamierzałem umieścić każdą klasę w oddzielnym pliku, jednak po pewnym czasie doszedłem do wniosku, że z tak zdefiniowanymi klasami jest to niemożliwe.

W tym miejscu dodaję na szybko napisaną wzmiankę o tym, jak działa limit czasowy bota, o czym wspomniane było w pliku readme. Jako, że nie mogłem sprawdzać, czy czas upłynął podczas wezwania rekurencyjnej funkcji minimax – nie było to korzystne dla złożoności bota

rozwiązanie przy wielu milionach wywołań naraz oraz że czas ruchu rośnie wykładniczo z każdym dodanym punktem do głębii, mogłaby wydarzyć się sytuacja, gdzie bot przez pierwsze kilka sekund, mieszcząc się w limicie, przemieli 6 pierwszych głębii i tuż przed jego końcem zacznie liczyć siódmą. Wtedy czas, który faktycznie upłynie przed ruchem jest niestety dużo wyższy, niż limit. Ze względu na to, bot przestaje iteracyjnie liczyć głębie, jeśli zostało mu mniej niż 3/10 limitu, ponieważ w większości przypadków i tak by się w nim nie zmieścił. Niestety jest to coś trudnego do sparametryzowania na wszystkich poziomach, więc tak, jak powiedziałem - limit czasowy jest czymś orientacyjnym, dokładnym tylko do pewnego stopnia.

Przechodząc do prawdziwie refleksyjnej części dokumentacji – chciałbym w tym miejscu powiedzieć, że mimo początkowych problemów i paniki związanej z małą ilością czasu na zrealizowanie projektu, naprawdę dobrze się przy nim bawiłem i jestem pewien, że w wolnym czasie będę kontynuował ulepszanie algorytmu grającego w grę, który okazał się dla mnie najciekawszą częścią zadania. Duma, którą poczułem, gdy po raz pierwszy pokonał on najpierw losowego bota, a później mnie, była nie do opisania. Szczerze powiedziawszy, czuję lekki niedosyt i nie jestem usatysfakcjonowany z uzyskanym wynikiem. Na poziomie głębii 8 przegrywam z nim większość gier, a miałem okres, w którym intensywnie interesowałem się szachami, więc ten klimat nie był mi kompletnie obcy, jednak czas, którego bot potrzebuje na odpowiedź to definitywnie coś do poprawy, nad czym popracuję więcej po oddaniu projektu. Algorytm służący do oceny pozycji jest teraz bardzo skromny, jednak gdy dodawałem do niego coś mającego za zadanie go poprawić w oddzielnym branchu, w którym funkcja do ewaluacji była przekazywana jako parametr przy tworzeniu instancji bota, okazywało się, że proste dążenie do przewagi materiałowej było skuteczniejsze. Być może wynika to z mojego płytkiego zrozumienia tej gry, mimo jej prostoty w porównaniu do szachów. Nie byłem pewien za co i ile punktów przyznawać w ewaluacji, więc końcowo nie była ona zbyt efektywna. Bot, który bonusowe ułamki punktów dostawał za średnie ,,wysunięcie" pionków na połowę przeciwnika (logicznie, są one bliżej stania się królem, więc są bardziej niebezpieczne), popełniał związane z tym błędy i pionki, które udało mu się tak wysunąć, były bardzo kruche. Jeszcze jedna uwaga odnośnie działania bota – czasami, gdy jest on w pełni przegranej pozycji, ma on możliwość ucieczki, jednak nie decyduje się na to, gdyż widzi, że za x ruchów nie będzie to miało znaczenia. Jest to pewna wada, pokazująca się dość rzadko, jednakże obecna i zależna od istoty algorytmu minimax i oceny sytuacji takiego bota. Podobne spostrzeżenia były wspomniane w jednym z podlinkowanych niżej przeze mnie źródeł.

4. **Źródła pomocne przy realizacji projektu**
   The ultimate introduction to Pygame
   Algorithms Explained – minimax and alpha-beta pruning
   Ta seria filmów związanych z implementacją warcabów w pythonie
   Raport z podobnego projektu porównujący ze sobą różne algorytmy i funkcje do ewaluacji
   Podobny opis problemu, jak w powyższym źródle – to stąd wziął się pomysł dania botowi limitu czasowego