HALMA

Bartosz Latosek

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Semestr zimowy 2020/2021

Projekt zrealizowany w ramach przedmiotu Podstawy Informatyki i Programowania

SPIS TREŚCI:

1.OPIS PROJEKTU I JEGO ROZWIĄZANIA

Moim zadaniem było stworzenie programu grającego w chińskie warcaby - halmę. Program miał obejmować 3 rodzaje rozgrywki: gracz kontra gracz, gracz kontra komputer i symulację gry komputera z komputerem. Program miał kontrolować poprawność wykonywanych ruchów.

Biblioteki użyte w programie:

> Pygame (Cały program praktycznie opiera się na bibliotece pygame)

Zasady gry w halmę, którymi się kierowałem:

> Rozgrywka toczy się na kwadratowej planszy 16 x 16

> Celem gry jest wypełnienie wszystkich wolnych pól w bazie przeciwnika swoimi pionkami

- Przeciwnikiem jest gracz posiadający bazę w przeciwległym narożniku planszy

> Wariant gry na 2 graczy obejmuje grę 19 pionkami i 19 pól bazy danego gracza, zaś jeśli zdecydujemy się na grę w 4 osoby każdy gracz ma do dyspozycji 13 pionków i odpowiadającą im bazę.

> Gracze poruszają się analogicznie do figury króla w szachach. Może również przeskakiwać nad dowolną ilością pionków (niezależnie czy należą one do przeciwnika, czy do gracza wykonującego ruch, o ile:

- Pole końca sekwencji skoków nie należy do żadnej z baz przeciwników postronnych

- Sekwencja skoków jest wykonana prawidłowo, to znaczy gracz przeskakując pionek ląduje na polu tuż za przeskakiwanym pionkiem, skąd ponownie może wykonać skok, o ile to możliwe

Program ostateczny znacząco różni się od tego, jak wyobrażałem go sobie na samym początku.

Zdecydowałem się na zebraniu wszystkich funkcji w przyjemnym dla oka użytkownika interfejsie graficznym, wyglądem przypominającym gry pokroju “8-bitówek”. Ponadto, zdecydowałem się na stworzenie prostego soundtracka do menu głównego, żeby jeszcze bardziej urozmaicić doświadczenia płynące z gry i sprawić, żeby cała koncepcja gry w halmę była nieco mniej nudna (stąd żartobliwy dopisek w menu głównym - ‘totally not borring’)

2. KLUCZOWE KLASY I FUNKCJE

1. Checker (checker.py)

Klasa ta jest odpowiedzialna za indywidualnego pionka i przechowuje wszystkie informacje go definiujące, takie jak kolor, pozycję na planszy, informacje o tym czy jest on aktualnie wybrany itp. Posiada też funkcję rysującą jednostkę pionka na planszy, dostosowywującą atrybuty pionka do jego wyglądu (kolor, status bazy, status home). Klasa w 100% pokryta testami.

2. Board (board.py)

Klasa ta jest główną jednostką wykonawczą rozgrywki. To ona zarządza pionkami na planszy, ich przemieszczaniem, przydziela je graczom na początku rozgrywki, zmianą statusu pionka, gdy spełnione są podane warunki. Ponadto, posiadając informacje o planszy logicznym wydaje się, by to właśnie ta klasa zwracała listę ruchów możliwych do wykonania przez dany pionek. Rekursywnie wylicza wszystkie dostępne ruchy i zwraca je w formie listy, którą następnie weryfikuje się czy wybrany przez gracza ruch jest legalny. Klasa Board jest też odpowiedzialna za odświeżanie wyglądu planszy w każdej klatce gry, rysuje tło, każdego z pionków i znaczniki legalnych ruchów. Ponadto nakłada animację na poruszanie pionkami. Klasa w 100% pokryta testami.

3. Game (game.py)

Ta klasa jest głównym koordynatorem wyglądu rozgrywki. Przyjmuje parametry, od których zależy min. Liczba graczy, tryb rozgrywki, trudność AI, występowanie animacji czy efektów dźwiękowych. Jest też odpowiedzialna za przebieg rozgrywki samej w sobie, w zależności od wybranego trybu. Klasa w 100% pokryta testami.

4. Player (game.py)

Prosta, nieskomplikowana klasa przechowująca podstawowe informacje o graczu, takie jak nazwa czy kontrolowany kolor. Klasa w 100% pokryta testami.

5. Bot (game.py)

Bot dziedziczy po klasie Player. Jest odpowiedzialny za wybranie najlepszego możliwego ruchu dla AI w danej rundzie. Działa obliczając odległości względne między obecnym położeniem każdego pionka, a destynacją. Następnie wybiera najoptymalniejszy ruch i implementuje go. Jeżeli wybrany pionek akurat znajduje się w bazie przeciwnika, to dodatkowo sprawdza, czy aby na pewno optymalnym będzie przesunięcie pionka, który już znajduje się w bazie. Klasa w całości testowana praktycznie, gdyż stworzenie testów jednostkowych byłoby wyczerpujące i nie przewidywałoby wszystkich możliwości rozgrywki.

6. EasyBot (game.py)

Jest to swojego rodzaju uproszczenie klasy Bot, będące pierwotnym algorytmem szukania najlepszego ruchu ulepszonym o wyszukiwanie najbardziej optymalnego ruchu dla wybranego pionka. Klasa różni się od klasy Bot tym, że pionek jest wybierany losowo, nie ważne czy znajduje się on już w bazie przeciwnika czy nie. Dla tak wybranego pionka dopiero wybierany jest najoptymalniejszy ruch. Podobnie jak klasa Bot, EasyBot była testowana praktycznie.

7. MainMenu (main_menu.py)

To klasa w całości odpowiedzialna za menu główne programu. Wyświetla animacje, zapisuje informacje o wybranych opcjach. Przygotowuje argumenty, które następnie zostaną przekazane do klasy Game w celu stworzenia wybranego modelu rozgrywki. Klasa testowana praktycznie ze względu na liczność funkcji z biblioteki Pygame.

8. Constants (constants.py)

To zbiór kluczowych zmiennych programu.

3.Instrukcja użytkowania

Do działania programu niezbędne jest zainstalowanie biblioteki pygame.

Aby uruchomić program należy uruchomić plik main_menu.py. Po uruchomieniu na ekranie pojawi się następujące okienko:

Domyślnie rozgrywka jest ustawiona na 2 osoby, gracz kontra gracz. Aby zmienić ustawienia należy nacisnąć ikonę koła zębatego.

W tym miejscu można zmienić interesujące nas opcje.

> Starting player: zmienia gracza rozpoczynającego rozgrywkę

> Sound Effects: Wycisza / odcisza muzykę w tle jak i dźwięki

przeskakiwania pionków

> Bot Difficulty: Ustawia trudność rozgrywki w grze z botem

> Tiny window: Ustala rozmiar okienka. W trybie tiny window

Animacje są niedostępne

> Players: Ustala liczbę graczy

> Mode: Ustala tryb rozgrywki, gdzie P oznacza gracza,

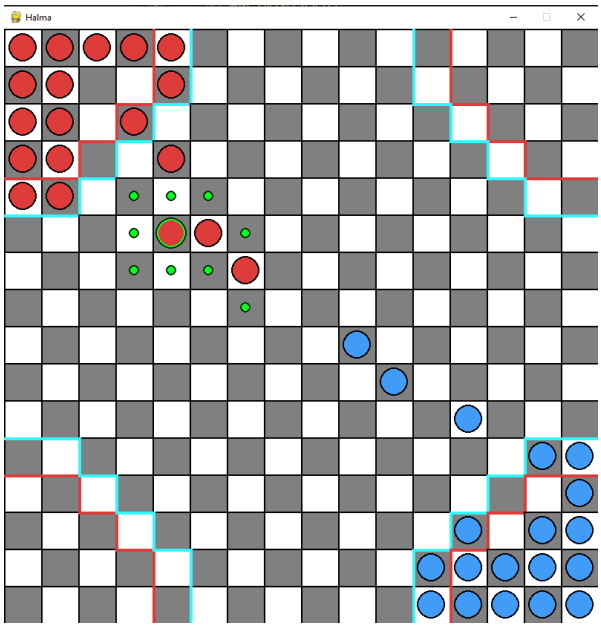
Natomiast Ai oznacza algorytm

> Animations: Włącza / wyłącza animacje

Aby rozpocząć rozgrywkę należy wcisnąć ponownie koło zębate (aby wpierw powrócić do menu głównego), a następnie wcisnąć przycisk PLAY

W zależności od wybranych opcji rozgrywka będzie przebiegała inaczej.

Kiedy nadejdzie tura gracza, należy kliknąć pionek, którym zamierzamy się przemieścić, a następnie wybrać pole z zielonym kółeczkiem, do którego możemy skoczyć.



Rozgrywka toczy się zgodnie z ruchem wskazówek zegara.

4. Testy Projektu

Testowałem jedynie funkcje i metody nie będące bezpośrednio związane z biblioteką pygame. Było to moje pierwsze doświadczenie z projektowaniem prostych gier komputerowych. Zdecydowałem się też na testowanie napisanych algorytmów w praktyce, pisanie testów jednostkowych do każdej możliwej sytuacji na planszy i sprawdzenie czy algorytm faktycznie wybiera najbardziej optymalny ruch byłoby bezcelowe. Funkcje graficzne takie jak rysowanie planszy czy pionków, również testowałem praktycznie. Reszta funkcji i metod klasowych jest w 100% pokryta testami jednostkowymi, znajdującymi się w plikach opisanych przedrostkiem test_ a następnie nazwą testowanej klasy.

5.Wnioski i obserwacje

Jak już wcześniej pisałem nigdy wcześniej nie miałem do czynienia z projektowaniem prostych gier komputerowych czy trywialnych algorytmów z nimi związanych.

1. Wniosek:

Stworzenie prostej gry za pomocą języka python i biblioteki pygame nie jest takie trudne jak początkowo mi się wydawało. Niemniej jednak stworzenie działającego programu było dla mnie wyzwaniem i poszerzyło zakres moich możliwości. Dodatkowo była to świetna praktyka dla projektowania obiektowego, którego nauczyłem się na przedmiocie PIPR.

2. Wniosek:

Drobny błąd w rozumowaniu, czy błędnie napisana funkcja mogą przełożyć się na dni wstrzymania pracy nad programem i wypełnione intensywnym debugowaniem w celu odnalezienia rozwiązania problemu. Dlatego tak istotne są testy jednostkowe i ich regularne pisanie jak i uruchamianie po każdej większej modyfikacji kodu.

3. Wniosek:

Zdalne repozytorium to bardzo dobre rozwiązanie zarówno na przechowywanie plików programu jak i backup w przypadku, gdyby po modyfikacji kodu przestał on działać poprawnie.