Prowadzący: dr inż. Rafał Biedrzycki

Dokumentacja końcowa

Projekt w ramach przedmiotu ZPR
Warcaby rosyjskie
Paweł Srokosz, Tomasz Nazarewicz, Paulina Pietrzyk

Opis zadania

Tematem projektu są warcaby w wariancie rosyjskim z interfejsem na przeglądarkę. Aplikacja zostanie zrealizowana w architekturze klient-serwer, gdzie rozgrywka będzie się toczyć między użytkownikiem a komputerem.

Zasady gry

Gra jest rozgrywana na planszy 8x8 pól pokolorowanych na przemian na kolor jasny i ciemny. Każdy gracz rozpoczyna grę z dwunastoma pionami (jeden koloru białego, drugi czarnego) ustawionymi na ciemniejszych polach planszy.

Pierwszy ruch wykonuje grający pionami białymi, po czym gracze wykonują na zmianę kolejne ruchy. Celem gry jest zbicie wszystkich pionów przeciwnika albo zablokowanie wszystkich, które pozostają na planszy, pozbawiając przeciwnika możliwości wykonania ruchu. Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne). Zbite piony są usuwane z planszy po zakończeniu ruchu. Piony mogą bić zarówno do przodu, jak i do tyłu. W jednym ruchu wolno wykonać więcej niż jedno bicie tym samym pionem, przeskakując przez kolejne piony przeciwnika. Bicia są obowiązkowe. Jednak jeśli istnieje kilka możliwych bić: wykonane może być dowolne.

Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, nawet w sytuacji, gdy zaszło to w trakcie bicia. Damki mogą poruszać się o dowolną ilość pól, zatrzymując się na wolnych polach. Bicie jest możliwe przez przeskoczenie pionu/damki przeciwnika, za którym znajduje się przynajmniej jedno wolne pole. Podczas bicia nie można przeskakiwać więcej niż jeden raz przez ten sam pion (damkę).

Funkcjonalności

Gra rozgrywa się z serwerem wyposażonym w mechanizm sztucznej inteligencji. Gracz ma możliwość wyboru jakimi pionkami chce grać. Czas rozgrywki jest odmierzany, a następnie uwzględniany w rankingu jeśli gracz zwycięży - udało się zrealizować funkcjonalności zaplanowane podczas tworzenia projektu aplikacji.

Techniczny opis projektu

Projekt składa się z dwóch części: serwera udostępniającego mechanizm sztucznej inteligencji i umożliwiający zdalną rozgrywkę, a także klienta działającego w środowisku przeglądarki i stanowiącego interfejs aplikacji.

Serwer

Serwer jest złożony z dwóch współpracujących ze sobą modułów: części komunikacyjnej i obliczeniowej.

Część komunikacyjna, napisana w języku Python, zawiera serwer WebSocket, który pośredniczy między zdalnym klientem, a algorytmem sztucznej inteligencji. W trakcie rozgrywki przede wszystkim przyjmuje żądania (ruchu) od zdalnego klienta (gracza), przetwarza je i przy pomocy części obliczeniowej generuje odpowiedź (zestawienie nowej gry, ruch serwera, informacja o przegranej/wygranej). Ponadto realizuje dodatkowe funkcjonalności, takie jak ranking, czy timer czasu rozgrywki.

Komunikacja odbywa się poprzez obiekty JSON.

Część obliczeniowa napisana jest w C++. Algorytm sztucznej inteligencji zbudowany jest na bazie algorytmu minimax z ulepszeniem alfabeta, o maksymalnie 10-poziomowym drzewie gry. Funkcja oceniająca powstała na bazie pracy Marcina Borkowskiego “Analiza algorytmów dla gier dwuosobowych”.

Obie części komunikują się przez interfejs klasy GameState części obliczeniowej. Udostępnia ona metody realizujące ruch gracza, żądające ruchu serwera i weryfikujące, czy któraś ze stron w wyniku ruchu nie wygrała.

Klient

Klient został zaprojektowany na bazie wzorca projektowego MVC. Kod napisany jest w języku TypeScript. Język ten jest kompilowany do JavaScript, który jest wykonywany przez przeglądarkę. Język TypeScript umożliwia programowanie zorientowane obiektowo i statyczne typowanie, co ułatwia pisanie aplikacji HTML5.

Model przechowuje lokalny stan rozgrywki. Udziela informacji o możliwych ruchach gracza, co pozwala na zablokowanie niedozwolonych posunięć z poziomu interfejsu. Z racji, iż wielokrotne bicia wysyłane są w jednym żądaniu ruchu, pozwala zweryfikować, czy bicie było finalne i wysłać ruch dopiero wtedy, gdy będzie kompletny. Ponadto przechowuje stan lokalnego timera, który jest synchronizowany z serwerem po zakończeniu rozgrywki.

Model jest aktywny i powiadamia widok o zmianie stanu planszy (widok jest obserwatorem modelu).

Widok jest jedną z bardziej skomplikowanych części klienta. Nad renderowaniem elementów graficznych i poprawnym rozmieszczeniem tekstur czuwa biblioteka pixi.js. Renderer opakowany jest w menedżer widoków, który umożliwia przełączanie między np. widokiem menu, a gry i pośrednio czuwa nad interaktywnością elementów (reakcją na kliknięcia myszy).

Widok jest obserwatorem modelu. JavaScript jest dość asynchroniczny, co prowadzi do wystąpienia zjawiska powszechnie znanego jako “callback hell”. Aktualizacja widoku wiąże się jednak z wykonaniem wielu synchronicznych czynności takich jak np. przemieszczenie pionka, usunięcie bitego pionka i zmiana pionka w damkę.

Problem eliminuje mechanizm kolejki czynności do wykonania (wzorzec komendy). Gdy widok otrzyma powiadomienie od modelu, aktualizuje swój stan i dodaje do kolejki odpowiednią czynność. Następnie kolejka jest uruchamiana. Czynności wykonywane są krokowo, przy rysowaniu kolejnej klatki (np. przemieszczenie pionka o jeden krok w kierunku docelowego miejsca). Gdy wykonywana czynność przy aktualizacji stwierdzi, że doszła do końca: uruchamia kolejną. Gdy kolejka już jest pusta, informowany jest kontroler, który wykonuje kolejne akcje.

Kontroler dzieli się na dwie części. Jedna część stanowi klienta WebSocket, zamienia akcje użytkownika w konkretne żądania i przetwarza odpowiedzi serwera, odpowiednio modyfikując stan aplikacji. Klient WebSocket komunikuje się z resztą aplikacji klienckiej poprzez metody głównego kontrolera aplikacji. Główny kontroler stanowi serce klienta. Zawiera on m.in. funkcje obsługi zdarzeń widoku i klienta sieciowego, które w uzgodnieniu ze stanem przechowywanym w modelu, wykonują zaprogramowane czynności, realizując funkcjonalność aplikacji.

Aplikacja kliencka udostępniana jest przeglądarce poprzez serwer HTTP - lighttpd.

Liczba linii kodu - statystyki

Podano liczbę linii kodu wraz z komentarzami i pustymi liniami (LOC - Physical Lines Of Code) i bez tych elementów (LLOC - Logical Lines Of Code)

Aplikacja kliencka (TypeScript):

LLOC: 1711 linii
LOC: 3014 linii

Serwer (Python i C++):

LLOC: 1186 linii
LOC: 1557 linii

Łącznie

LLOC: 2897 linii
LOC: 4571 linii

Sposób testowania

Przed połączeniem klienta z serwerem część obliczeniowa serwera (napisana w języku C++) została wielokrotnie przetestowana poprzez interfejs konsolowy - dzięki zapisowi wykonywanych przez gracza ruchów możliwe było odtworzenie ich sekwencji prowadzącej do błędnego zachowania lub interesującego stanu planszy - wybory sztucznej inteligencji są deterministyczne (powtarzalne dla tych samych sekwencji). W kodzie podczas testowania została umieszczona odpowiednia ilość asercji sprawdzających takie rzeczy jak np. spójność danych i poprawność zakresów zmiennych, przez co wiadomo, że na żadnym poziomie drzewa gry nie pojawiły się błedy tego typu.

Klient początkowo był testowany oddzielnie, użytkownik mógł poruszać pionkami obydwu kolorów. To pozwalało na przetestowanie podstawowych funkcjonalności modelu i widoku klienta takich jak np. wymuszanie i wykonywanie bić czy poprawne poruszanie się damki.

Po podłączeniu klienta z serwerem przetestowane zostały takie sytuacje jak utrata łączności klienta z serwerem, zapis dobrego czasu na listę rankignową, oraz sama rozgrywka - wielokrotne bicia, bicia podczas których pion staje się damką, przegrana (bądź wygrana) w skutek braku pionów którymi można się poruszyć itp.

Decyzje podejmowane przez sztuczną inteligencję okazały się być na wystarczająco wysokim poziomie - nie zawsze prowadzą do przegranej użytkownika, natomiast powodują, że rozgrywka jest odpowiednio ciekawa.

Napotkane problemy

Największy problem w tworzeniu aplikacji sprawiła implementacja i testowanie części serwera odpowiadającej za działanie algorytmu sztucznej inteligencji. Opiera się on na wielokrotnej rekurencji, co bardzo utrudniło wyszukiwanie i naprawę potencjalnych błędów.

Jako, że logika wielu różnych operacji wykonywanych na pionach i damkach jest podobna, w pewnym momencie problem zaczęła stanowić występująca redundancja kodu. Udało się ją jednak ograniczyć.

Czas przewidziany podczas tworzenia dokumentacji wstępnej został bardzo niedoszacowany - głównie przez nieuwzględnienie wystarczającej jego ilości na testowanie aplikacji.

Zadania do wykonania: czas przewidziany/czas poświęcony na projekt

Realizowane zadanie	Czas realizacji (w godzinach)
1. Zaprojektowanie modelu rozgrywki i protokołu wymiany danych między klientem, a serwerem	
Opracowanie algorytmu wykorzystywanego przez sztuczną inteligencję (serwer)	4 / 5
Zaprojektowanie struktur danych służących do wymiany informacji między klientem a serwerem	2 / 2
Implementacja algorytmu “sztucznej inteligencji” z użyciem pseudokodu	4 / 4
2. Zadania związane z tworzeniem klienta	
Stworzenie szkieletu aplikacji zgodnie ze wzorcem Model-View-Controller	4 / 4
Opracowanie interfejsu i wewnętrznych mechanizmów modelu, zgodnego z zakładaną logiką gry	3 / 4
Napisanie kodu dla modelu w postaci zestawu klas w języku TypeScript	3 / 4
Testy jednostkowe modelu	1 / 2
Zaprojektowanie widoku	
Projekt interfejsu graficznego (stworzenie grafik - sprite’ów, ogólny zarys, pozycja poszczególnych elementów)	3 / 3
Opracowanie logicznej struktury widoku z perspektywy kodu (podział na podwidoki: ładowania gry, menu głównego, widoku rozgrywki)	2 / 2
Napisanie kodu dla widoku, z użyciem biblioteki pixi.js (renderera HTML5 dla grafiki 2D)	4 / 6
Testy jednostkowe poszczególnych elementów widoku	2 / 2
Powiązanie widoku i modelu	
Opracowanie kontrolera sprzęgającego widok z modelem, powiązanie widoku z modelem poprzez funkcje obsługi zdarzeń.	3 / 5
Implementacja klienta sieciowego (element kontrolera), wykorzystującego uprzednio zaprojektowany protokół do komunikacji z serwerem.	4 / 4
Testowanie rozgrywki: czy serwer współpracuje poprawnie z klientem i	3 / 4

odwrotnie. Wymuszanie sytuacji wyjątkowych, np. rozłączenia w trakcie rozgrywki.	
Testy funkcjonowania na popularnych przeglądarkach.	1 / 3
3. Zadania związane z tworzeniem serwera	
Opracowanie sposobu przechowywania danych (ranking)	1 /1
Opracowanie wewnętrznych mechanizmów modelu, zgodnego z zakładaną logiką gry (np. kontrola stanu rozgrywki, czasu jej trwania itd.)	4 /3
Implementacja komunikacji z klientem, ww. mechanizmów, algorytmów rozgrywki	4 /5
Testowanie działania serwera (np. poprzez wysyłanie JSONów przez klienta WebSocket, rozegranie gry poprzez https://www.websocket.org/echo.html), wymuszanie sytuacji wyjątkowych	4 /12
4. Dokumentacja projektu	
Opracowanie dokumentacji końcowej	1 / 1
Suma	**57 / 76**