10.06.2014 r.

Przedmiot: ZPR

Projekt: gra karciana Fantasy przez sieć

Skład zespołu:

Mateusz Prewysz Kwinto

Marta Jolanta Łępicka

Paweł Szymczyk

Dokumentacja końcowa

W ramach projektu zrealizowano grę karcianą, opartą na mechanice gier takich jak Hearthstone.

Przebieg rozgrywki:

Gracze logują się do gry za pomocą unikatowego loginu. Po zalogowaniu otrzymują talię kart dwóch typów: zaklęcie oraz sojusznik. Talia nie jest do wglądu - karty do ręki dobierane są według kolejności wystąpienia w talii. W swojej turze użytkownik ma trzy możliwe ruchy: dobranie kart do ręki z talii (maksymalnie pięć kart w ręce), rzucenie karty z ręki na stół (maksymalnie pięć kart na stole) oraz przeprowadzenie ataku. Przeprowadzenie ataku jest możliwe tylko przez karty znajdujące się na stole. Aby to zrobić należy wybrać kartę a następnie cel ataku – kartę przeciwnika lub samego przeciwnika. Jeżeli cel ataku był kartą, jej dalsze zachowanie określone jest przez jej typ: jeżeli jest zaklęciem zniknie, jeżeli sojusznikiem – sojusznik otrzyma zadaną część obrażeń ale nie zniknie, póki jego punkty życia będą większe od zera. W przypadku gdy celem ataku jest przeciwnik – otrzymuje on normalne obrażenia i odliczana jest mu odpowiednią ilość punktów życia.

Udało się zrealizować:

Strukturę gry podzielono według wzorca: Model-View-Controller.

-w modelu gry:

Udało się zrealizować rozróżnienie kart na dwa typy: sojusznicy oraz zaklęcia i uzależnienie od nadanego typu ich działania, żywotności. Model odpowiednio reaguje na komunikat

otrzymywany od serwera. Każda istotna zmiana w modelu (przeciwnik odniósł obrażenia, przełożono kartę na stół itp) adnotowana jest w odpowiednim momencie i zapisywana do makiety danych, która wysyłana jest do serwera. Gra kończy się gdy któryś z graczy będzie miał poniżej zera punktów życia.

-w serwerze (kontroler) gry:

Serwer realizuje komunikację między modelem gry a widokiem, przesyłając odpowiednie komunikaty i nadając rysowanie widoku. Zarządza zdarzeniami i zapewnia, że model otrzyma odpowiednie informacje w celu rozegrania tury, a widok w celu narysowania aktualnego stanu gry.

-w widoku(klience) gry:

Schemat działania klienta: Na samym początku wyświetlane jest okienko logowania. Gracz wpisuje swój nick i po kliknięciu przycisku Start łączy się z serwerem za pomocą webSocketa. Serwer jest odpytywany o stan gry co parę sekund i na podstawie odpowiedzi rysowana jest plansza gry. Dodatkowe komunikaty są wysyłane przy wykonaniu ruchu przez określonego gracza - wyłożenie karty albo atak kartą.

Plany na przyszłość:

W przyszłości planowane jest umożliwienie zaklęciom czarów leczących czy zwiększających atak. Rozróżnienie typu bohatera i uzależnienie od tego możliwych posiadanych zaklęć, punktów życia czy many. Planuje się również wprowadzenie bazy kart – praktycznie każda karta charakteryzowałaby się unikatowym działaniem (różna siła ataku, koszt many itp.).

Sposób realizacji i opis podstawowych funkcji:

-model gry:
Ważniejsze funkcje na najwyższym poziomie obiektów: Wrapper:
void readCommunication(Communication) – odpowiedzialna za odczytywanie komunikatu otrzymywanego od serwera I nadawnie odpowiednich stanów gry, tury. Rozpoznaje również którego gracza jest aktualnie tura w celu określenia aktualnego gracza – gracza rozgrywającego.

Mockup startGame(Communication) – odpowiedzialna za rozruch i start gry. Utworzenie modelu oraz stołu dla graczy o danych id oraz zaalokowanie pamięci. Dba o odpowiedni zapis danych do makiety wysyłanej do serwera.

Mockup makeMove(Communication) – odpowiedzialna za rozegranie odpowiedniej akcji w zależności od rodzaju ruchu gracza. Tutaj rozróżniane są trzy możliwości: rzucanie kart na stół, dobieranie kart z talii do ręki oraz atakowanie przeciwnika. Dba o odpowiedni zapis danych do makiety wysyłanej do serwera.

Mockup endGame() - odpowiedzialna za prawidłowe zakończenie gry, zwolnienie pamięci oraz zaktualizowanie makiety danych z odpowiednimi stanami.

Mockup update(Communication) – metoda zbierająca wszystkie powyższe metody i wołająca je w odpowiednich momentach (w zależności od komunikatu: stanu tury, stanu gry, rodzaju ruchu gracza).

Szczegółowa dokumentacja w formacie html.

-serwer gry:

Serwer gry zrealizowano w pythonie. Przesyłane komunikaty parsowane są do jsona dla widoku, oraz do odpowiednich typów dla modelu.

-klient gry (widok):
JS klasy:
Card (name, image, description, attack, health) - ogólna struktura przechowująca dane karty JS
Funkcje/metody/eventy:
$("#cardsPlayerBoard").on('click', '.card', function()) - kliknięcie na kartę na stole. Powoduje jej zaznaczenie, by zaatakować kartę przeciwnika.
$("#cardsOpponentBoard").on('click', '.card', function()) - wybranie karty przeciwnika do zaatakowania, o ile jakaś z własnych kart została wybrana
$("#opponentImage").click(function()) - wybranie przeciwnika jako cel ataku
$('#playerHand').on('click', '.card', function()) - wybranie karty z ręki do wyłożenia na stół
$("#start").click(function ()) - rozpoczęcie połączenia z socketami na serwerze po wciśnięciu przycisku start
callback - wysyłany callback do serwera, aktualnie co 3 sekundy z prośbą o informację o stanie gry
ws.onmessage - 'narysowanie' całej planszy gry na podstawie otrzymanych informacji od serwera