Maciej Bogusławski, Hubert Kaczyński, Amadeusz Lewandowski, Bartosz Żelazko

PAP 2024Z ZESPÓŁ 16 ETAP 3

Gra strategiczna „Habemus Rex!”

Realizacja wymagań funkcjonalnych

W ramach etapu trzeciego projektu, każde z 20 pierwotnych wymagań funkcjonalnych ukończone zostało w 100%.

Wymaganie	Stan	Co zostało zrobione?
Implementacja funkcjonalności interaktywnej mapy	100%	Mapa regionu, możliwość przesuwania i przybliżania/oddalania mapy, zjawiska pogodowe, blokowanie wyjścia poza mapę, reprezentacja pól na mapie, wybór pól na mapie
Implementacja ekranu startowego gry	100%	Ekran startowy otwierający grę, możliwość startu gry lub natychmiastowego wyjścia
Implementacja mechaniki pól	100%	Wczytywanie nazwy, opisu i obrazu wybranego miasta, wczytywanie budynków wybudowanych w danej sesji gry dla danego miasta oraz ich ikon, implementacja mechaniki wykrywania przynależności pól do gracza bądź przeciwników i funkcjonalności
Implementacja funkcjonalności wizualnego menu budowy budynków	100%	Lista budynków dostępnych w grze wraz z ich ikonami, nazwami, opisami oraz statystykami, interaktywny przycisk informujący o możliwości wybudowania danego budynku lub o istnieniu danego budynku w danym mieście, natychmiastowe odświeżanie listy budynków po wybudowaniu budynku
Implementacja wizualnych elementów odpowiedzialnych za prezentację statystyk	100%	Pasek zasobów z ikonami reprezentującymi dane surowce oraz informacjami o obecnym stanie surowców i turowym balansie ich przychodów i wydatków, elementy prezentujące statystyki budynków oraz jednostek, przyciski budowania dopasowujące się do obecnej możliwości zakupienia budynku przez gracza, możliwość podglądu opisu budynków po najechaniu na nie myszą
Implementacja funkcjonalności wizualnego menu rekrutowania jednostek	100%	Menu armii wywoływane z paska zasobów, podgląd ikon jednostek, ich nazw, opisów, kosztu, utrzymania, ataku, obrony, przycisk umożliwiający rekrutację oraz zwolnienie danych jednostek w armii, informacja o posiadaniu jednostek danego typu w armii
Kreacja oprawy graficznej rozgrywki	100%	Ikony, miniatury i obrazy, tło ekranu startowego, animacja pogody na mapie, animacja przycisku następnej tury, animacje przycisków po najechaniu myszą, graficzne wykończenia pasków, menu i paneli, dobór czcionek, efekty cienia i poświaty wokół tekstu oraz obrazów
Implementacja funkcjonalności zapisu i odczytu stanu gry	100%	Pełna implementacja klasy wczytującej i zapisującej dane dynamiczne do bazy oraz odpowiednio filtrującej i zwracającej potrzebne dane w zależności od wartości parametrów. Zapisywanie i odczytywanie budynków oraz armii gracza i przeciwników, przynależności pól, odbytych bitew i stanu zasobów graczy.
Implementacja logiki budynków	100%	Dodanie budynków dostępnych w grze wraz z ich statystykami, nazwami, opisami i miniaturami do bazy danych, filtrowanie wybudowanych na danym terytorium budynków w panelu miasta oraz panelu konstrukcji budynków, mechanika poboru zasobów gracza po wybudowaniu budynku, mechanika analizowania możliwości wybudowania budynku na danym terytorium
Implementacja logiki jednostek bojowych	100%	Dodanie jednostek bojowych dostępnych w grze wraz z ich statystykami, nazwami, opisami, analizowanie posiadania przez gracza armii danej jednostek oraz możliwości rekrutacji nowych jednostek bojowych. Implementacja funkcjonalności analizujących możliwość zatrudnienia danej jednostki oraz mechanik automatycznie aktualizujących ilość zasobów aktualnie posiadanych przez gracza lub przeciwnika oraz aktualizujących stan dochodów oraz kosztów zasobów po zrekrutowaniu, zwolnieniu, eliminacji bądź dezercji. Wprowadzenie mechaniki odpowiedzialnej za dezercję jednostek gracza w przypadku braku możliwości zaspokojenia kosztów armii.
Implementacja logiki zasobów	100%	Dodanie zasobów dostępnych w grze wraz z ich nazwami, opisami i ikonami do bazy danych oraz tabel wyznaczających koszt, produkcję oraz utrzymanie budynków i jednostek, przechowywanie stanu zasobów wszystkich graczy, wyznaczanie bieżącego bilansu dochodu i kosztu turowego dla każdego surowca każdego z graczy, algorytmy zarządzania zasobami przez graczy wirtualnych
Implementacja mechaniki upływu czasu	100%	Licznik tur, przycisk następnej tury, mechanizm odświeżającego się zegara systemowego, wykrywanie wystąpienia sytuacji bliskiego zakończenia gry i informowanie gracza o takiej sytuacji, zaimplementowanie algorytmów zwiększania swojej siły militarnej i potencjału gospodarczego przeciwników wraz z upływem czasu, wykrywanie końca gry
Implementacja funkcjonalności związanych z turowością gry	100%	Dodanie tabel wyznaczających koszt, produkcję i utrzymanie budynków i jednostkę co turę do bazy danych, przechowywanie informacji o obecnej turze, stworzenie klasy filtrującej koszt, produkcję i utrzymanie danego budynku, mechanizm zmiany tury obejmujący aktualizację stanu pól, dochodów oraz kosztów, stanu zasobów, armii i budowli, implementacja algorytmów zarządzających ruchami i podejmowania decyzji dotyczących rozgrywki przeciwników między turami gracza, wykrywanie sytuacji krytycznych
Implementacja logiki działania przeciwników	100%	Dodanie pojęcia frakcji oraz wirtualnych graczy w formie tabel do bazy danych, implementacja funkcjonalności przynależności pól do poszczególnych graczy, implementacja algorytmów zarządzania konstrukcjami, stanem zasobów oraz budową armii na podstawie algorytmów decyzyjnych wyznaczających współczynniki oparte o koszty jednostki i obecny stan zasobów, wzbogacanych o czynnik losowy
Implementacja funkcjonalności odpowiedzialnej za logikę walki	100%	Dodanie jednostek bojowych dostępnych w grze wraz ze statystykami ich siły i zdrowia do bazy danych, implementacja mechanizmów kontrolujących liczność jednostek danego typu w armii. Stworzenie algorytmów odpowiedzialnych za obliczanie wyniku walki na podstawie siły bojowej armii atakującej i siły defensywnej obrońcy. Dodanie elementów odpowiedzialnych za usuwanie jednostek utraconych w walce, liczba jednostek utraconych jest zależna od wyniku walki. W przypadku zwycięstwa gracza pole które zostało podbite przechodzi w ręce zwycięzcy, umożliwiając mu na budowanie w nim budynków, a także czerpanie korzyści z elementów, które znajdowały się na polu przed walką.
Implementacja reagowania na zdarzenia awaryjne	100%	Kontrolowanie możliwości zakupienia danego budynku przez gracza poprzez porównywanie kosztów budynku i aktualnych zasobów gracza, implementacja funkcjonalności wyznaczającej obecny bilans kosztów i dochodów z budynków oraz armii, implementacja funkcjonalności kontrolujących obecny stan zasobów graczy oraz przeciwników oraz aktualizujących ten stan na koniec tury, implementacja algorytmów kontrolujących zachowanie przeciwników oraz ich decyzje w oparciu o możliwość wykonania danego działania, implementacja algorytmów wykrywających sytuacje, w których bilans zasobów gracza lub przeciwnika na koniec tury nie jest w stanie zaspokoić potrzeb swoich budynków lub armii, implementacja funkcjonalności wyłączającej budynki, których koszty nie mogą zostać pokryte, implementacja funkcjonalności eliminującej odpowiednie jednostki armii w przypadku, w którym jej koszty nie mogą zostać pokryte, mechanizm informowania gracza o automatycznej reakcji systemu na sytuację, w której zasoby gracza są niewystarczające
Implementacja funkcjonalności wizualnego przedst. walki	100%	Implementacja elementów graficznych odpowiedzialnych za informowanie przeciwnika o zdarzeniach, implementacja menu podboju, w którym obecne są sumaryczne informacje dotyczące sił gracza w porównaniu z

		siłami przeciwnika. Siły przeciwnika objęte są efektem „mgły wojny”, która ukrywa to jakimi dokładnie siłami dysponuje przeciwnik, podając jedynie estymacje. Po rozwikłaniu bitwy wyświetlane jest okno podsumowujące wyniki i straty.
Implementacja mechaniki powiadomień	100%	Implementacja funkcjonalności wykrywających sytuacje wymagających wywołania powiadomienia, wywoływanie mechanizmów sprawdzających konieczność pokazania powiadomień i wywołujących powiadomienia pod koniec tury, implementacja klasy nieinwazyjnych powiadomień, implementacja graficznych elementów interfejsu powiadomień oraz animacji wywołujących oraz chowających powiadomienia
Stworzenie mechaniki punktacji oraz zwycięstwa	100%	Implementacja menu punktacji otwieranego po naciśnięciu przycisku obecnego na panelu górnym gry, zawierających informacje o posiadanych punktach każdej frakcji. Implementacja mechaniki naliczania punktów zgodnie z algorytmem uwzględniającym posiadane przez danego gracza bądź przeciwnika terytoria oraz budynki. Dodanie mechanik natychmiastowego aktualizowania punktacji wraz ze zdobywanymi terytoriami i budynkami. Implementacja mechanik wykrywania zwycięstwa oraz informowania o takiej sytuacji gracza. Implementacja menu zwycięstwa, umożliwiającego kontynuację gry lub uruchomienie menu pozwalającego na zapis gry.
Stworzenie ekranu pomocy	100%	Implementacja menu pomocniczego opisującego zasady i funkcjonalności gry w formie przewijalnej listy zaopatrzonej w szczegółowe opisy oraz pomocnicze grafiki, implementacja przycisków uruchamiających ekran pomocniczy oraz elementów graficznych interfejsu ekranu pomocy.

Lista wymagań funkcjonalnych wraz z ich pełnymi opisami znajduje się w katalogu docs w ramach sprawozdania z Etapu I projektu.

W ramach funkcjonalności dodatkowych przygotowany został serwer REST API stworzony za pomocą technologii Spring Boot + Spring Security, implementacja połączenia klienta gry i serwera w całości stanowić będzie przedmiot dalszych prac nad projektem we własnym zakresie. Serwer obsługuje rejestrację i logowanie użytkownika oraz wysłanie i pobieranie danych do i od klienta (tylko wymagane funkcje dla klienta gry). Wszystkie endpointy serwera są zabezpieczone (oprócz oczywiście logowania, rejestracji) za pomocą mechanizmu JWT (Json Web Token) oraz refresh tokenów trzymanych w bazie danych, wykorzystywanych do ich odświeżania. W obecnym stanie, w kliencie połączonym z serwerem można się tylko zalogować, zarejestrować (za pomocą nowego menu wbudowanego w klienta) i rozpocząć nową grę, która pobierze część potrzebnych danych do uruchomienia pojedynczej sesji (danych statycznych, czyli danych, które dla każdej rozgrywki są takie same). Integracji na poziomie danych dynamicznych niestety nie udało się zaimplementować ze względu na wcześniej wspomniane komplikacje więc w konsekwencji uruchomienie sesji w połączeniu klient-serwer nie jest możliwe. Serwer jest uniwersalny więc po odpowiednim rozwinięciu można by go również wykorzystać np. do aplikacji webowej, którą można by było w jakiś sposób zintegrować z grą np. poprzez panel administracyjny, w którym administrator mógłby zarządzać zarejestrowanymi użytkownikami. Najnowsza wersja serwera znajduje się w branchu "fullstack" i jest modułem o nazwie "server".

Opis zrealizowanej aplikacji

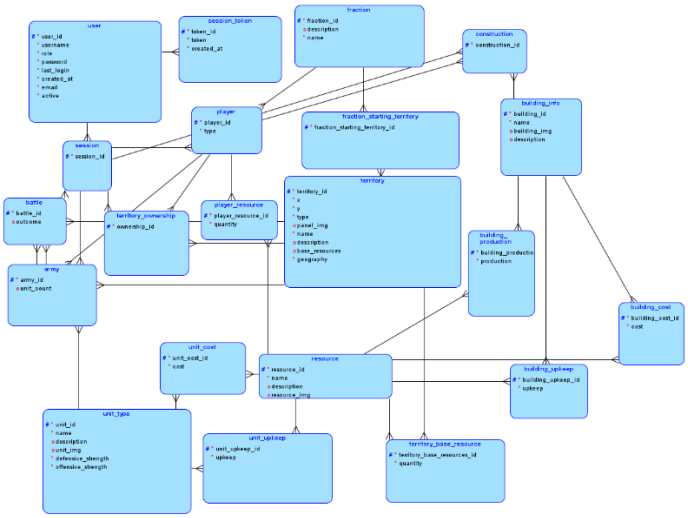
Gra „Habemus Rex!” stworzona została na bazie technologii Java 21 oraz JavaFX w wersji 21.0.5. W celu uruchomienia bazy danych, ustanowienia z nią połączenia oraz interakcji z bazą gra korzysta z narzędzia Docker oraz biblioteki Hibernate. W celu automatyzacji budowania projektu gra korzysta również z narzędzia Maven.

Kod gry znajduje się w branchu „game”. W pakiecie aplikacji w folderze cachedData znajdują klasy odpowiedzialne za wczytywanie i zapisywanie danych statycznych i dynamicznych gry. W folderze entities znajdują się klasy ORM, odpowiedzialne za mapowanie elementów gry, umożliwiając interakcje z bazą danych. W folderze media znajduje się klasa odpowiedzialna za elementy audio. W folderze enemyLogic znajdują się klasy odpowiadające za implementację logiki związanej ze sterowaniem zachowaniem przeciwników. W folderze ui znajdują się klasy reprezentujące elementy interfejsu graficznego gry. Klasa Main odpowiada za uruchomienie gry. W katalogu test/java znajdują się testy jednostkowe oraz integracyjne testujące zaimplementowane funkcjonalności, mechaniki oraz połączenie z bazą danych.

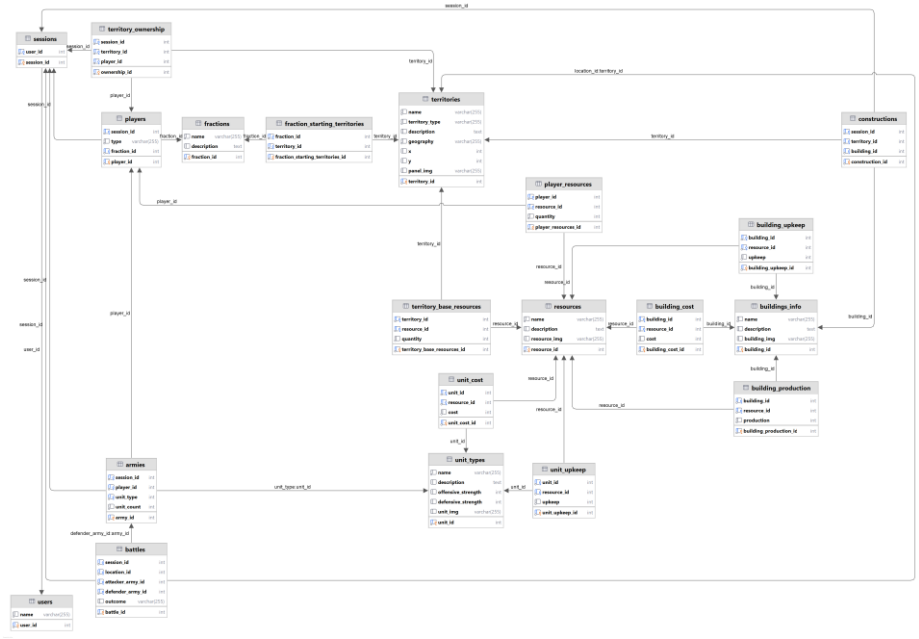
Plik .jar w branchu main skompilowany został tak, aby do jego uruchomienia nie było potrzebne manualne zainstalowanie biblioteki JavaFX. Ten sam plik gry uruchomić można zarówno w systemie Windows, jak i systemie Linux. Preferowanym sposobem instalacji i uruchomienia jest uruchomienie skryptu z katalogu głównego – sposób ten został przedstawiony w filmie prezentującym prototyp w ramach Etapu II oraz w README w katalogu głównym.

Schemat bazy danych

Baza danych gry „Habemus Rex!” składa się z 20 tabel. Poniżej znajduje się diagram ER.



Poniżej znajduje się diagram fizyczny.



Oba diagramy znajdują się w pełnej rozdzielczości w katalogu docs/etap III w branchu main.