POZNAN UNIVERSITY OF TECHNOLOGY

Tymoteusz Bleja
Paweł Husak
Patryk Imosa
Magalena Łątkowska

Internetowa gra edukacyjna ucząca podstaw pracy z programem git

Praca inżynierska

Supervisor: dr hab. inż. Marek Andrzej Wojciechowski

Poznań, 2017

Spis treści

Rozdział 1

Wstęp

1.1 Zasady odnośnie pisania pracy

- Piszemy w formie bezosobowej. Można też niby w 1.os liczby mnogiej czyli Źrobiliśmy...", ale niektórzy akceptują tylko bezosobową, czyli Źrobiono...".
- słów niepolskich - „Nie można więc bezpośrednio w tekście używać słów angielskich. Jeżeli już — powinny być wyróżnione kursywą" - niektórzy podobno bardzo hejcą za angielskie słowa niestety.
- Jak chodzi o bibliografię, to w wolnej chwili dołączcie linki do stron z jakich korzystaliście, ja to potem ogarnę i zapiszę w takiej formie jak trzeba. Ale spokojnie, bo robienie bibliografii raczej zostawię na koniec.
- Ogólnie nie przejmujcie się strukturą, formatowaniem czy innymi formalnymi bzdetami, ja potem będę to ogarniać żeby było wg zasad więc nie traćcie czasu na ogarnianie takich rzeczy.

Między innymi: skąd wgl pomysł - bo Git jest super i konieczny a ciężko się go samemu nauczyć, nauka z wielu źródeł jest chujowa, większość ma tylko blade pojęcie a potem idzie do pracy i dupa - co z tego że nauczyli się na studiach programować jak nie potrafią korzystać z Gita i współpracować z zespołem

Cel i zakres pracy

cel - nauka fajna łatwa i przyjemna, oraz praktyczna, obycie z typowymi scenariuszami jakie mogą być potrzebne w pracy

Coś o tym dlaczego akurat przeglądarkowa gra, czemu z grafiką 3D itp.

Cytat z karty pracy : Zapoznanie się z systemem Git. Opracowanie koncepcji interaktywnego samouczka do nauki podstaw korzystania z systemu Git. Opracowanie architektury systemu. Implementacja i testowanie systemu. Przygotowanie dokumentacji technicznej i użytkowej.

Rozdział 2

Podstawy teoretyczne

2.1 Sysytem kontroli wersji Git

Może rozdział powinien się nazywać „Wstęp teoretyczny", albo od razu „System kontroli wersji Git" lub samo „Git", i wtedy bez tej podsekcji - i tak w rozdziale nie może być tylko jedna

Krótki opis systemów kontroli wersji, do czego służą i jakie dają korzyści. Wyjaśnienie dlaczego Git się wyróżnia, co ma szczególnego i bardziej obszerny jego opis.

Jakaś subsekcja o tym dlaczego programiści powinny go znać i potrafić używać jako narzędzia w pracy, do czego im się przyda i że Git często nie jest możliwością, lecz koniecznością.

Rozdział 3

Projekt

3.1 Założenia

Najważniejsze cele, co ma spełniać gra, czego ma nauczyć i w jaki sposób

3.2 Przebieg gry

Krótki opis rozgrywki i elementów interfejsu (może screeny? nie jestem pewna czy już w sekcji „projekt" czy dopiero w sekcji „implementacja")

3.3 Zadania

Ta sekcja nie tyle ma być o zadaniach w sensie Taskach, taskGraph.json itp, ale o zadaniach w kontekście co ma zrobić użytkownik, czego i w jaki sposób ma się przy tym nauczyć.

3.3.1 Konstrukcja zadania

Pojedyncze zadanie reprezentowane jest w postaci obiektu. W każdym z nich znajdują się informacje o tytule zadania, jego opis, który służy ukazaniu użytkownikowi celu oraz problemu jaki należy rozwiązać w danym zadaniu. Może on być na przykład przedstawiony w postaci zlecenia dodania jakiejś funkcjonalności lub naprawienia błędu.

Aby nagradzać użytkownika punktami za szybko wykonane zadanie zdecydowano się na określenie czasu na wykonanie zadania. Czas jest obliczany w momencie dodawania nowego zadania na podstawie ilości dotychczas wykonany zadań, a także wartości minimalnej oraz maksymalnej czasu w jakim można to zadanie wykonać.

Ponad to zadania posiadają listę następników, tj. identyfikatory zadań, które mogą wystąpić po aktualnym zadaniu, gdyż hierarchia zadań ułożona jest w postaci grafu skierowanego. Zdecydowano się na taką strukturą ze względu na to, że daje ona możliwość sekwencjonowania zadań, a zarazem pozwala na element losowości, gdyż kolejne zadania wybierane są w sposób losowy z puli z następników aktualnego zadania. Dzięki takiej reprezentacji nigdy nie zdarzy się sytuacja, w której następnym zadaniem będzie zadanie nieadekwatne do aktualnego stanu repozytorium.

Zadania zawierają również listę kroków, które należy spełnić, aby problem został uznany za rozwiązany. Dzięki podziałowi zadania na kroki użytkownik uczy się jak należy postępować i jakich użyć komend, aby sprostać postawionemu zadaniu. Ma to również na celu automatyzowanie zachowania użytkownika w prawdziwych przypadkach, z którymi może się spotkać w domu lub pracy. Na każdy krok składa się opis co należy w nim zrealizować. W systemie git możliwe jest wykonanie pewnych akcji przy użyciu różnych komend czy przełączników. W tym celu każdy krok zawiera listę komend, które użytkownik może zastosować, by uznać, że krok został wykonany poprawnie. Krok jest równoznaczny z wykonaniem pojedynczej komendy w systemie git.

Dla każdego kroku określamy listę tagów. Przez tagi rozumiemy etykiety jakie nadajemy krokom w celu określenia zagadnienia jakiego uczą. Etykiety te są związane z wykonywanymi komendami. Jednym z przykładowych tagów może być 'commit', który jest nadawany danemu krokowi, gdy realizujemy w danym kroku akcję zatwierdzania zmian w naszym repozytorium. W zadaniach istnieje wiele tagów takich jak: 'branch', 'checkout', 'merge', 'pull', 'push', itd. Gdy użytkownik poprawnie wykona dany krok to zwiększany jest wskaźnik powodzenia dla wszystkich etykiet przypisanych do danego kroku. Przechowywane są informacje o etykietach w postaci ilości wykonanych kroków z daną etykietą oraz ilość tych pozytywnych wywołań. W ten sposób można określić w czym użytkownik sobie gorzej radzi. Ta informacja jest brana pod uwagę przy losowaniu kolejnych zadań, gdyż dla możliwych następników wyliczana jest waga na podstawie etykiet jakie mają w swoich krokach. Obliczenie wagi dla zadania polega na zsumowaniu kwadratów ilorazu wartości 1 oraz wyznaczonego wskaźnika dla kolejnych tagów w krokach zadania, będącego ilorazem poprawnie wykonanych kroków z całkowitą liczbą prób dla tej etykiety. Dzięki takiemu zabiegowi zwiększane jest prawdopodobieństwo wylosowania kolejnego zadania posiadającego etykiety w swoich krokach, z którymi użytkownik miał większy problem. Przykład kroku, w którym wykorzystywana jest więcej niż jedna etykieta może stanowić wykonanie komendy pozwalającej na jednoczesne utworzenie gałęzi w repozytorium oraz przejście na nią, w tym przypadku przypisanymi wartościami będą 'branch' oraz 'checkout'.

Kolejnymi elementami będącymi częścią kroków w zadaniu są typ oraz dane, które niesie ze sobą dany krok. Typ pozwala na określenie jaką akcję należy wykonać po stronie grafiki 3D, by odwzorować aktualny stan repozytorium. Dane wykorzystywane są w różny sposób. Dostarczają informację np. o nazwie 'commita' czy 'brancha'. Mogą też określać jakie pliki powinny zostać dodane lub usunięte z repozytorium przy wykonywaniu akcji 'pull'. Innym razem niosą informację czy 'checkout' powinien być realizowany na 'branch', a może na konkretny 'commit'.

3.3.2 Podpowiedzi/Pomoc

Nie wiem jak nazwać tę subsekcję, bo HelpDrawer to nie do końca pomoc czy podpowiedzi, tylko content do nauczenia, nie umiem tego ładnie nazwać na razie.

Coś że gdy w zadaniu jest krok z nową komendą to wyskakuje pomoc i co w niej jest itp.

3.3.3 Punkty za rozwiązanie

W projekcie wprowadzono elementy gamifikacji np. poprzez punkty za rozwiązanie zadań. Został dla nich stworzony komponent, który reaguje na zakończenie zadania. Przy wybieraniu zadania jako kolejne ustawiana jest wartość nagrody za poprawne wykonanie całego zadania. Wartość ta obliczana jest na podstawie minimalnego czasu przeznaczonego na dane zadanie pomnożonego przez ilość wykonanych zadań oraz liczbę 10. Co daje nam możliwość premiowania zadań, które są są wykonywane głębiej naszego drzewa zadań oraz mają być wykonane szybciej niż inne zdania. Gdy zadanie zostaje wykonane pobierany jest czas wykonania zadania. Jeżeli zadanie zostaje wykonane po przekroczeniu określonego na zadanie czasu użytkownik nie otrzymuje punktów. Natomiast jeżeli użytkownik zmieścił się w czasie, obliczona wcześniej nagroda zostaje pomnożona przez stosunek czasu, który został do czasu przeznaczonego na zadanie. Całość jest zaokrąglana w górę do liczby całkowitej. Obliczona wartość jest dodawana do całkowitej ilości punktów jaką dotychczas zdobył gracz.

Elementy takie jak punkty powodują, że gra wciąga użytkownika. Z każdym rozegraniem, użytkownik stara się pobić swój dotychczasowy wynik. W ten sposób co raz to szybciej i pewniej będzie wykonywał zadania oraz komendy. Dodatkowo punkty wprowadzają elementy rywalizacji pomiędzy użytkownikami. Użytkownicy będą starać się zajmować najwyższe miejsce w rankingu, co powoduję, że będą wykorzystywać aplikacje w celu uzyskania poprawy wyniku.

Rozdział 4

Implementacja

4.1 Wykorzystane technologie

4.1.1 Redux

krótki opis i zalety, czemu wybraliśmy

4.1.2 React

4.1.3 Babylon.js

Babylon.js jest otwartą biblioteką WebGL napisaną w TypeScript wykorzystywaną przede wszystkim do tworzenia gier wideo w przeglądarkach. Pierwsza odsłona została wydana w 2013 roku. Głównymi twórcami są David Cotuhe oraz David Rousset. Jako, że jest silnikiem 3D posiada wspaniałe narzędzia do tworzenia, wyświetlania i teksturowania szkieletów w przestrzeni. Przez to, że kierowana jest głównie do twórców gier posiada wiele dodatkowych funkcji jak generowanie krawędzi, gotowych obiektów ułatwiających pracę programiście typu tworzenia obiektu na podstawie mapy wysokości. Ponadto zapewnia natywną detekcję kolizji, grawitację sceny, czy wbudowane kamery np. FollowCamera - kamera śledząca, automatycznie podążająca za obiektem.

Rozważana była jeszcze inna biblioteka 3D, a mianowicie Three.js wydana w 2009 roku, również oparta na podstawie webGL. Po zapoznaniu i przetestowaniu obu silników wybór padł na Babylon.js. Głównymi czynnikami skłaniającymi nas w stronę Babylon'a była płynność, którą zapewniał w przeciwieństwie do Three.js oraz prostota użycia. Przy renderowaniu tego samego, wybranego obiektu Babylon.js spisywał się lepiej o kilka klatek. Poza tym Babylon.js jest dobrze udokumentowany, posiada bogatą bazę poradników oraz społeczność wykorzystujące tę bibliotekę jest duża i stale rośnie.

4.1.4 Pozostałe?

4.2 GraphCreator

Jak opisano w rozdziale JAKIMS zadania zawierają wiele niezbędnych informacji. Dodatkowo tworzą strukturę drzewiastą. Wymagało to zastosowania wygodnego formatu do

przechowywania tych danych. Zdecydowano się na format JSON, który to idealnie pasuję do tego zadania oraz jest łatwy w odczytywaniu i modyfikowaniu w języku JavaScript. Aplikacja GITar Hero umożliwia przetwarzanie dużych scenariuszy z wieloma zadaniami, dlatego ręczne tworzenie takiego scenariusza byłoby uciążliwe i wygenerowany graf mógłby zawierać wielę nieporządanych błędów. Dodawanie lub modyfikacja zadań wymagała by znaczących nakładów pracy. Z tego powodu zdecydowano się na stworzenie narzędzia z graficznym interfejsem użytkownika, w którym to można by było w prosty i wygodny sposób stworzyć należyty graf zadań, a następnie zapisać go w formacie JSON. Postanowiono poszukać gotowych rozwiązań w internecie. Najlepszą aplikacją okazała się "directed-graph-creator"użytkownika cjrd znalezioną na stronie https://github.com/cjrd/directed-graph-creator (NIE WIEM JAK Z LINKAMI) napisaną w języku JavaScript. Umożliwia ona stworzenie w łatwy sposób grafu skierowanego i zapisanie go w formacie JSON. Oprogramowanie jest na licencji MIT/X dzięki czumu istnieje nieograniczone prawo do używania, kopiowania, modyfikowania i rozpowszechniania go. Jedynym wymogiem jest, by we wszystkich wersjach zachowano warunki licencyjne i informacje o autorze. Narzędzie korzysta z popularnej biblioteki D3.js, która to pozwala tworzyć dynamiczne i interaktywne wizualizacje danych w przeglądarkach internetowych. Każdy węzeł grafu mógł przechowywać tylko pojedyńczy ciąg znaków. Potrzebna była zatem modyfikacja umożliwiająca zapisanie w pojedyńczym węźle wszystkich niezbędnych informacji, które powinny znaleźć się w zadaniu. Wymagało to zmiany struktury węzła oraz dodanie dodatkowego panelu do wypełniania danych zadania. Aplikacja napisana jest w czystym JavaScripcie bez użycia dodatkowych frameworków, a do tego całość zapisana jest w pojednyńczym pliku co utrudniało wprowadzanie nowej funkcjonalności. Postanowiono nie modyfikować struktury aplikacji i dalszą część napisać również w czystym JavaScripcie. Dodano jedynie bibliotekę jQuery umożliwiającą łatwiejsze zarządzanie elementami drzewa DOM. Związku z tym, że jest to aplikacja webowa potrzebny był serwer HTTP. Użyto do tego celu środowiska Node.js, który umożliwia stworzenie wysoce skalowalnych aplikacji internetowych. Największą zmianą było dodanie panelu bocznego umożliwiającego wypełnianie oraz podgląd danych zadania. Po kliknięciu na węzeł grafu, reprezentujący pojedyńcze zadanie, na bocznym banelu zostają wyświetlone wszystkie jego informacje takie jak tytuł, opis i czasy wykonania, które można w łatwy sposób modyfikować. Można również w wygodny sposób stworzyć listę kroków składających się na wykonanie zadania. Aby dodać jeden z nich należy wybrać jego typ i kliknąć przycisk "Dodaj", a następnie wypełnić pola opisujące dany krok. Można podać takie atrybuty jak opis, listę komend spełniających dany krok, tagi opisujące wykonane czynności jak i dodatkowe parametry, które są charakterystyczne dla różnych typów. (MOZE JAKIES ZDJECIE?) Stworzony graf zadań można zapisać w łatwy sposób do foramtu JSON. Aby to zrobić wystarczy kliknąć pierwszy przycisk w lewym dolnym rogu ekranu. GraphCreator wygeneruję strukturę grafu zrozumiałą dla aplikacji GITar Hero, określi zadanie inicjujące rozkgrywke i zapiszę wszystko do pliku taskGraph.json. Zapisane grafy można wczytać ponownie do aplikacji naciskając drugi przycisk w lewym dolnym rogu ekranu i wybierając plik do wczytania.

Operation: (NIE WIEM CZY DODAC TO DO TEKSTU CZY ZROBIC LISTE OPERACJI)

* drag/scroll to translate/zoom the graph * shift-click on graph to create a node * shift-click on a node and then drag to another node to connect them with a directed edge

* click on node or edge and press delete button to delete * click on node to edit it

4.3 Komponenty/elementy(?) aplikacji

4.3.1 Lista zadań

Screen jakiś, jak działa, o implementacji

4.3.2 Konsola

Screen jakiś, jak działa, o implementacji

4.3.3 Pomoc (HelpDrawer)

Screen jakiś, jak działa, o implementacji

4.3.4 Drzewko repo

Screen jakiś, jak działa, o implementacji

4.3.5 Canvas

Tutaj krótko co to, że wyłapuje zmiany stanu i jak wpisano dobrą komendę to przekazuje do Engine3D który jest odpowiedzialny za grafikę i robi co ma się stać.

4.4 Grafika 3D

Tu będzie sporo, do tego stopnie sporo, że nie wiem jeszcze jak to zaplanować i porozdzielać, względem czego.

Czy np podsekcje takie jak: repo3d - gałęzie, commity, co to są i jak powstają na akcje, o ich teksturze, obramowaniu, w tym o solidExplode ground - co to, jak działa itp particle w tle (wg elementów aplikacji)

Czy może raczej podsekcje wg 'elementów' Babylona: Meshe, SolidParticle, Particle, Materiały, Tekstury, Shadery

Rozdział 5

Podsumowanie

Dodatek A

Przewodnik użytkownika

Praktyczne info dla opornego użytkownika, jak ma korzystać, między innymi że jest opcja scrolla aby oddalić, jakie przyciski do obsługi helpa itp. itd., krótki opis fragmentu rozgrywki co się dzieje po czym i dlaczego i jak ma na to reagować użytkownik i takie tam.