Web Diplomacy

Design Document

12/6/2017

Version 3.0

Diplomats

Connor Wright

Sean Kallaher

Hugh McGough

Course: CptS 322 - Software Engineering Principles I

Instructor: Sakire Arslan Ay

TABLE OF CONTENTS

I. Introduction

This document is intended to illustrate our process for implementing web-based Diplomacy and describe the tools and procedures used therein. Diplomacy is a strategic board game of wits, cunning, negotiation, and deceit. This project is aimed at providing a platform to play the game of Diplomacy remotely with their friends.

In Section II, the overall architecture of the project is described.

In Section III, further details of each subsystem are provided.

In Section IV, the testing methods for the project are outlined.

In Section V, we provide useful references related to the project.

Document Revision History

Rev	1.0	2017-10-16	Initial version
Rev	2.0	2017-11-5	Iteration 2, update testing specifications
Rev	3.0	2017-12-6	Iteration 3,

II. Architecture Design

II.1. Overview

The major subsystems of this project include the User Interface, Database, and Game Engine. The relations between these subsystems are described in Appendix A. The Game Engine processes information from the UI and in the Database to determine the results of the players actions. The User Interface takes information from the user and sends it to the game engine using http requests. The database stores game information such as the map unit position and control points. The Database communicates with the Game Engine thorough queries. We reduced coupling having a linear form of communication. Every subsystem has at most two other subsystems it is talking to. Cohesion was reduced by giving each subsystem a clear task that is its only job. The model we are applying to our program is the Model View Controller. The User utilises the User Interface to send orders to the Game Engine (controller). The Game Engine then processes the orders and updates the Database (model). After updating the Database the Game Engine then updates the User Interface (view).

III. Design Details

III.1. Subsystem Design

This section provides more detail about each subsystem:

III.1.1. User Interface

The User Interface uses a combination of HTML, Javascript, and CSS to display and receive information about the game from the user. Communications between the User

Interface and the Game Engine use HTTP requests through Flask. The types of communication are defined as follows.

1. send order (POST): This will send an order to the game engine
    a. URL: /api/send_order
        i. {"select":select, "action":action, "target":target, "attack":attack};
    b. Response: 200 if valid, 418 if invalid
        i.
2. Confirm orders (POST): confirms the orders of the player.
    a. URL: /api/confirm_order
        i. JSON: N/A
    b. Response: 200 if valid, 418 if invalid
        i. {"teams": [{"army":["Lugo","IrishSea","London"],"navy":["NorthSea","Wales"],"score":"25","name":"batman"} ,{"army":["Paris","Nile","Kiev"],"navy":["AtlanticOcean","Wales"],"score":"25","name":"notbatman"}]};
3. load game (GET): loads the inital game data
    a. URL: /api/get_game
        i. JSON: NA
    b. Response: 200 if valid, 418 if invalid
        i. Same as confirm orders

III.1.2. Game Engine

The Game Engine is written in Python and utilizes Flask for handling HTTP requests as well as Psycopg to generate PostgreSQL queries. The communications between the Game Engine and the User Interface are outlined above. Internally, the Game Engine processes requests from the User Interface and determines the outcome of a myriad of actions. The most important piece of the Game Engine is be the action resolution algorithm which determines the resulting game state given a set of actions from all players. The game engine requires use of functions that

- Get all of the attack orders of a specified game
- Get the unit associated with an order
- Get all of the defense orders of a specified game
- Get the order from a given unit
- Get the unit occupying a location
- Get all of the orders that are attacking a specified location
- Get all of the orders that are convoys in a specified game
- Test if a location is empty or not
- Add orders to the database for a specific game

- Update locations of units in the database for a specific game
- Clear all orders in the database for a specific game
- Initialize the data in the database

III.1.3. Database

The Database uses a PostgreSQL database to store all of the critical information required for the Game Engine to operate. SQL queries will be used to access and modify this data from the Game Engine. The Database will be used as a representation of the current state of all games as well as a method to validate orders the users make.

III.2. Data design

Appendix B contains a schema diagram outlining the tables used by the Database subsystem. Internally, the Game Engine will rely heavily upon the Database as its primary data structure and will not utilize any other data structures. Appendix C contains the UML class diagram for the game engine. Appendix D contains the UML class diagram for the client side.

III.3. User Interface Design

User Interfaces have been simplified to include the game board. The user interface uses html, css and javascript. Liberal amounts of jQuery to make the program adaptable. There can now be up to 16 players in the gameboard. The Game page contains a map a sidebar and a bottom bar. The bottom bar contains the teams. The side bar contains inputs to use troops. The map contains all the countries. The map will display where all of the troops are as well as which countries are owned by which teams.

IV. Testing Plan

To test the Game Engine, we planned on using Python's unittest library. This would have provided an easy to use testing framework for the backend Python code in the form of unit tests for modules such as the order resolution algorithm. This could have also been easily expanded to perform API testing. Unfortunately we ran out of time for testing so this is how we would have implemented testing.

I. Unit Testing

We would have written automated testing for the following:

1. Order resolution
2. Order validation
3. Various functions that will be used to get valid entries for an order. For example getting valid locations that can be supported, or getting valid locations to move to.
4. Checking if two locations are neighbors.
5. Checking if two locations share a neighbor.
6. Adding a user to a game
7. Committing all of a team's orders
8. Removing a unit from a game

9. Relocating a unit in a game
10. Adding a unit to a game
11. Getting the origin of an order
12. Getting the target of an order
13. Getting the type of order
14. Getting the Secondary target of an order
15. Getting all the orders that attack a specified location
16. Getting all the orders that are Defending any location
17. Getting all of the orders that are a convoy

II. Functional Testing

We manually tested that a user is be able to perform the following

1. Add a new unit to one of their supply centers
2. Relocate a unit to a neighboring empty location or an empty supply center that they own
3. Join a game
4. Enter a valid order for their units
5. Submit their orders to the server
6. Leave a game

III. UI Testing

We manually tested that the ui is navigable and that it displays according to the specification.

V. References

The Rules of Diplomacy, Stephen Majesk, http://faculty.washington.edu/majeski/426/sim1.html

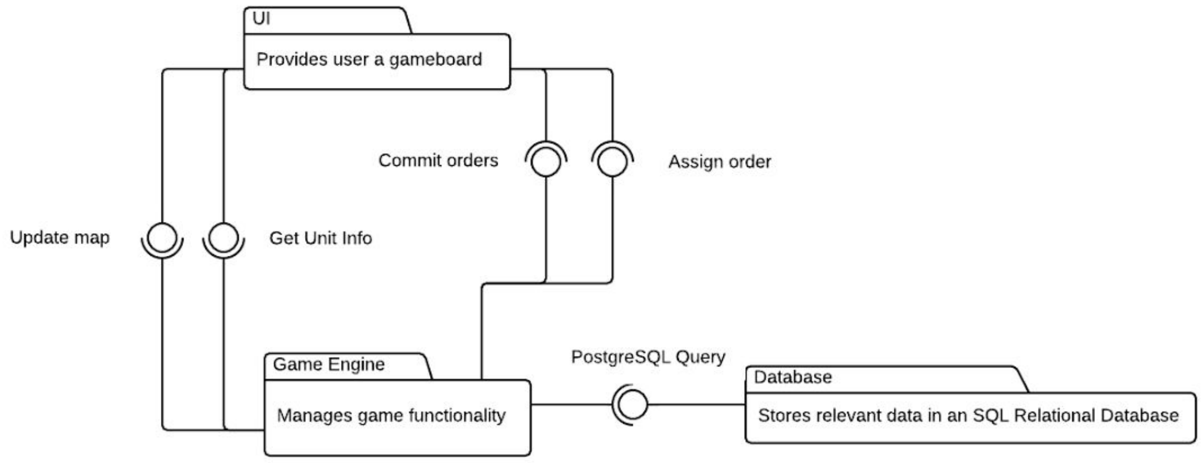
PostgreSQL 10.0 Documentation, https://www.postgresql.org/docs/10/static/index.html

Psycopg Documentation, http://initd.org/psycopg/docs/

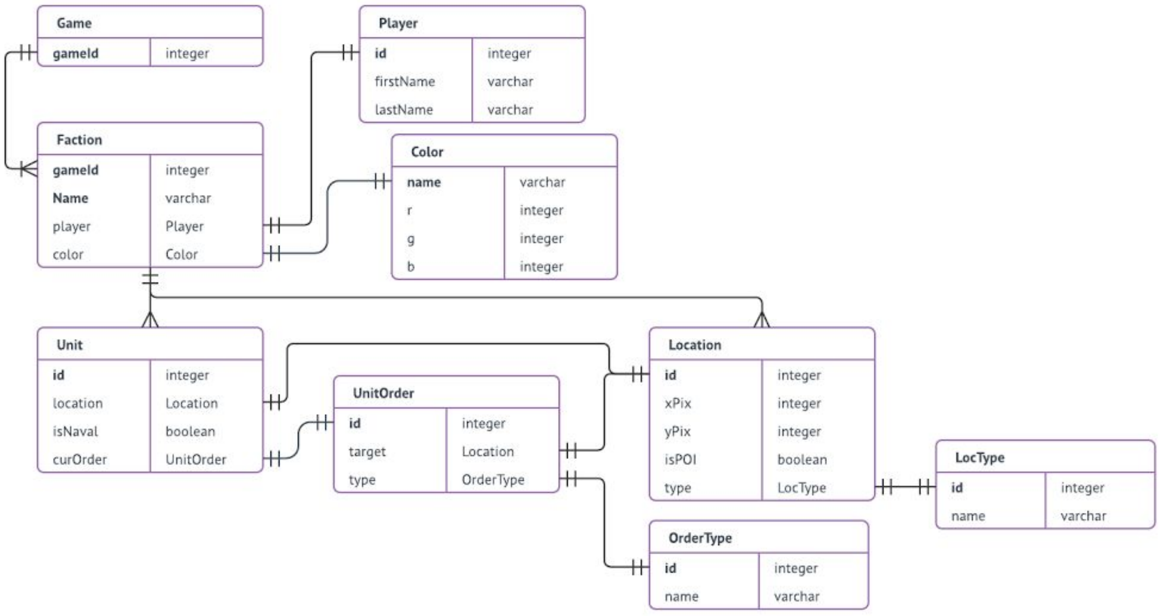
Python Unittest API, https://docs.python.org/3/library/unittest.html

jQuery Documentation, https://api.jquery.com

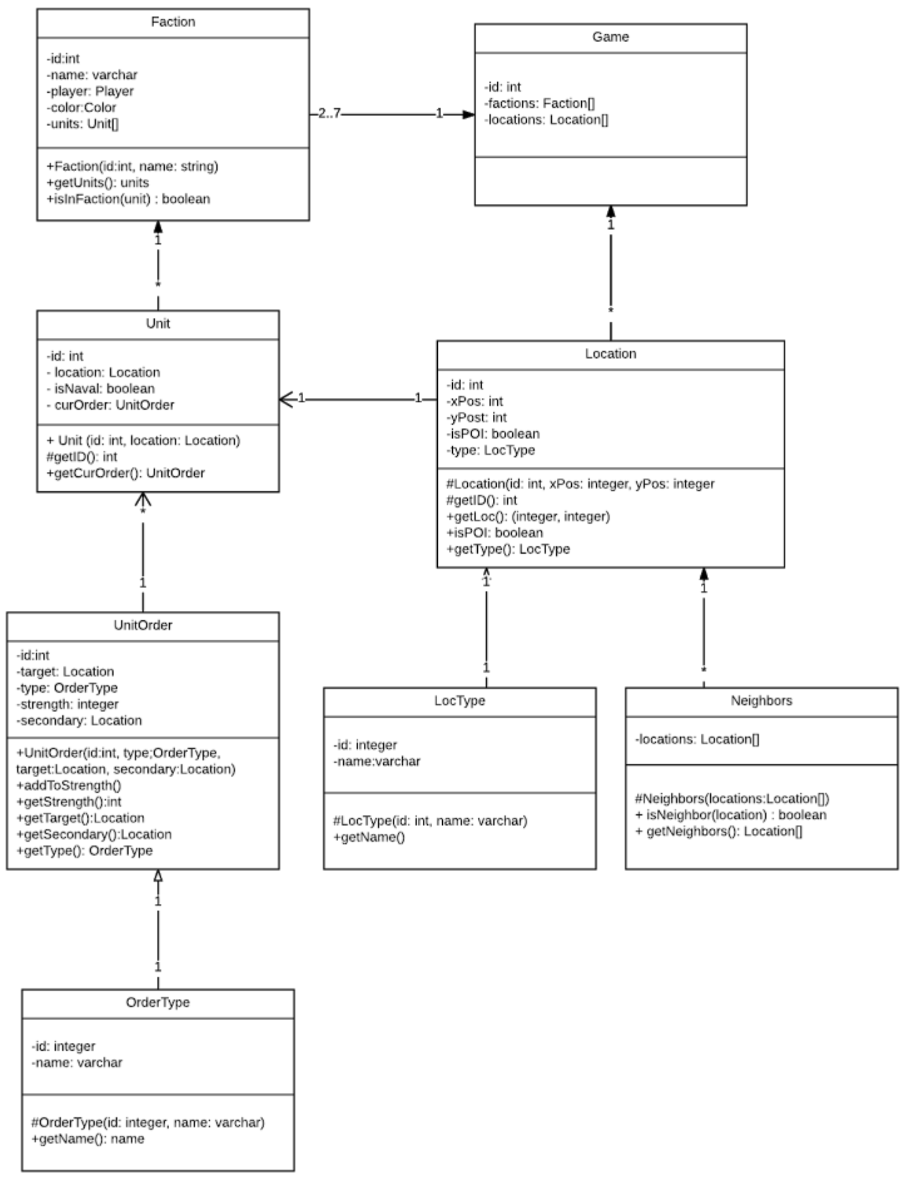
VI. Appendix A: UML Subsystem Block Diagram


UI
Provides user a gameboard
Commit orders
Assign order
Update map
Get Unit Info
Game Engine
Manages game functionality
PostgreSQL Query
Database
Stores relevant data in an SQL Relational Database

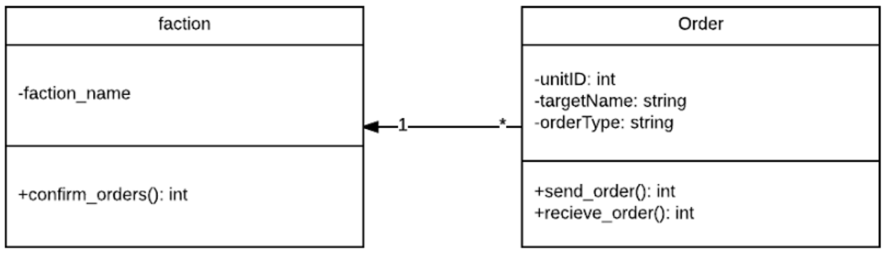
VII. Appendix B: SQL Schema



VIII. Appendix C: Game Engine UML Class Diagram



IX. Appendix D: Client Side UML Class Diagram


faction
-faction_name
+confirm_orders(): int
1
*
Order
-unitID: int
-targetName: string
-orderType: string
+send_order(): int
+recieve_order(): int