INTELL – The Craft of Intelligence Game

University of Kentucky

Department of Computer Science

Senior Design Spring 2016

April 28, 2016

Table of Contents:

1. Personnel
2. Disclaimer
3. Abstract
4. Introduction
   1. Customer and Intended Audience
   2. Need for Product
   3. Product Description
5. Product Requirements
   1. Game Interface
   2. Scenario Editor Interface
   3. Backend
   4. Apportioning of Requirements
6. Product Planning
   1. Effort Estimation
   2. Schedule and Milestones
   3. Platforms, Tools, and Languages
7. Design
   1. Environment
   2. Diagrams
      1. Use Case Diagram
      2. Architecture Diagram
      3. State Transition Diagram
   3. Design Considerations
   4. Formal Development Environment
8. Implementation
   1. Issues
   2. Design Changes
   3. Limitations
   4. Testing
9. Future Enhancements/Maintenance
   1. User Interface
   2. Move to Larger Server
   3. Add Scenario Rating System
   4. Add Advanced Snippet Generation
   5. Test User Friendliness in Depth
10. Conclusions
11. References
12. User Manual and Installation Guide
    1. Acquisition
    2. Requirements
       1. Environment
    3. Dependencies
    4. Instructions
       1. Installation
       2. Configuration
       3. Usage
    5. Acquiring Dependencies
       1. Python 3
       2. Django 1.9+
       3. django-crontab
       4. django-jenkins
       5. django-casper
       6. SQLite3
       7. Jenkins (optional)
13. Personnel

Christina Campbell

Braden Fullington

Casey O’Kane

Dylan Wright

1. Disclaimer

This project has been designed and implemented as a part of the requirements for CS-499 Senior Design Project for Spring 2016 semester. While the authors make every effort to deliver a high quality product, we do not guarantee that our products are free from defects. Our software is provided "as is," and you use the software at your own risk.

We make no warranties as to performance, merchantability, fitness for a particular purpose, or any other warranties whether expressed or implied.

No oral or written communication from or information provided by the authors or the University of Kentucky shall create a warranty.

Under no circumstances shall the authors or the University of Kentucky be liable for direct, indirect, special, incidental, or consequential damages resulting from the use, misuse, or inability to use this software, even if the authors or the University of Kentucky have been advised of the possibility of such damages.

1. Abstract

INTELL - The Craft of Intelligence is a multiplayer, turn oriented, strategy/mystery game that mimics the real life challenges of intelligence gathering and threat analysis. Each instance of the game is based on a scenario that follows the progress of a terroristic threat. The object of the game is to apprehend the villain of the scenario before they have a chance to execute their plans. Players must determine whether or not the snippets of information they receive are related to the scenario, which they are able to do by testing snippets for validity and also by investigating scenario characters and locations. Players also have the option to thwart one another by anonymously creating misinformation and attempting to execute other players’ agents. It is up to the players to determine which strategies and techniques will prove most useful in their attempts to apprehend the villain.

1. Introduction
   1. Customer and Intended Audience

The INTELL game concept was created by Tony Elam, and was originally played via email during a course taught at Rice University. Our team was tasked by Mr. Elam to create a web-based product that promotes ease of use and access for users interested in playing INTELL.

* 1. Need for Product

The email form of INTELL was popular among those who played, but gameplay was difficult and inconvenient. Players had to agree upon times to submit actions to the moderator, and the moderator had to find time to organize players’ choices/progress and email updates to all players. This process quickly proved to be inefficient and a hindrance to gameplay, which prompted Mr. Elam to consider removing the moderator role in exchange for an electronic, web-based version of the game.

* 1. Product Description

The game was developed to be accessed from a browser, thus allowing players the option to play when they please and removing the need for a moderator. The browser will supply a user interface, which displays all game updates to players and acknowledges all desired game actions from players. In addition to improving the platform, our team developed a database that reads in scenarios and traces player progress as it changes throughout the game instance. The browser based platform also includes a scenario editor, which allows authors and editors to create and edit their own scenarios to be played.

1. Product Requirements

Full requirements, complying with IEEE standard, for the project can be viewed on the project website at: http://intellproject.com/~dylan/INTELL/documents/requirements.shtml

General requirements for the project are listed below:

* 1. Game Interface

The game interface will be used by players. It will communicate with the backend system to retrieve scenario information. This system must be responsive, visually attractive, and intuitive. These requirements are described below.

**Responsive** This interface must respond to user actions within reasonable time frames. Where appropriate, this interface should indicate an operation is pending with appropriate messages and loading indicators. All user actions should have a visual response by the interface.

**Attractive** This interface will be used by players of the game so it must be the most polished. Care must be taken during implementation to ensure that components are visually appealing and complementary.

**Intuitive** This interface must be easy to use by players of all skill levels. Users should be able to perform core gameplay activities without navigating deeply into menus. Where appropriate, dialogs should be used to guide users in use of the interface.

* 1. Scenario Editor Interface

The scenario editor will be used by scenario authors. It will communicate with the backend system to retrieve and store scenario information. This system must be responsive and intuitive. These requirements are described below.

**Responsive** This interface must respond to user actions within reasonable time frames. Where appropriate, this interface should indicate an operation is pending with appropriate messages and loading indicators. All user actions should have a visual response by the interface.

**Intuitive** This interface must be easy to use by authors of all skill levels. Users should be able to create scenarios without navigating deeply into menus where appropriate dialogs and instruction text should be used to guide users in use of the interface.

* 1. Backend

The interfaces will use a common backend. The backend will contain the database used to store game information. The backend must be responsive and secure.

**Responsive** This system must respond to front end requests in a reasonable time frame. This system must be capable of handling requests from multiple front end systems simultaneously.

**Secure** This system must be secure from un-authorized access. Only those with proper credentials are allowed to directly access the database. Client systems may access the database through sanitized and front end generated requests. If user accounts are implemented user data must be safeguarded from un-authorized access and use.

* 1. Apportioning of Requirements:

The following features are listed in order of descending priority with some overlapping priority:

1. Backend Schema – Create database to hold scenario information.
2. Scenario Editor Tools – Implementation of add, edit, and delete functionality for different characters, locations, and events.
3. Scenario Editor Interface Skeleton – Basic user interface for accessing Scenario Editor.
4. Backend Game Logic – Controls game instances, processes user actions, and determines user interactions with the scenario and one another.
5. Backend Snippet Generation Using Author Defined Attributes – Generates snippets for presentation during gameplay.
6. Game Interface Skeleton – Basic user interface for accessing Game Instances.
7. Improved/Complete Scenario Editor Interface – A user interface that allows full access to all features of the Scenario Editor.
8. Improved Game Interface (able to play most of game) – A semi-complete user interface for accessing Game Instances.
9. Complete Game Interface (able to play and finish entire game) – A user interface that allows full access to all features of gameplay.
10. Polished Game Interface (beautification) – A themed, fully functional user interface for Game Instances.
11. Polished Scenario Editor Interface– A themed, fully functional user interface for the Scenario Editor.
12. Product Planning
    1. Effort Estimation

When making initial effort estimates, our team decided to model our estimates by using story points and calculating effort based on time. Our team estimated that all aspects of our project would take about 35 days’ worth of coding/work. Many modules of this project were created and edited concurrently, but our original estimates were largely underestimated. Our time estimates can be shown below with the initial estimation in parentheses and the actual value to the right:

* Editor
  + Editor HTML (0.5 days) 2 days
  + Editor logic (10 days) 21 days
* Game
  + Game HTML (1 day) 3 days
  + Game front end logic (10 days) 21 days
  + Back end generates snippets for the turn/back end determines which user gets which snippets for the current turn = (4 days) 21 days
  + Back end receives actions for all users and processes these actions and then calls the generate new snippet operation = (7 days) 21 days

Our team believes that our underestimations of time and effort can be attributed to the fact that none of us had ever worked on a project of this scale before. Additionally, our team had to take time to learn the languages and framework used to implement the project. This learning curve undoubtedly increased the amount of time necessary to spend creating the application.

As it currently stands, the project totals 11,312 lines of code.

* 1. Schedule and Milestones

Our initial schedule was agreed upon on February 7. Some items and dates have been added/changed since. Our final schedule appears below:

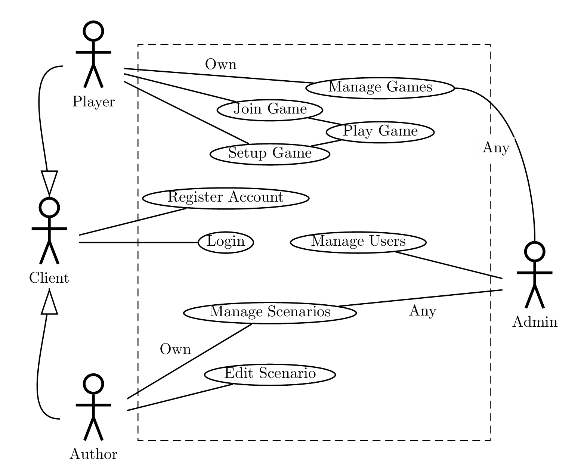
|  |  |
| --- | --- |
| Date | Goal |
| 02/12 | Web page with requirements due |
| 02/22 | Initial design draft |
| 02/29 | Formalize design |
| 03/03 | Review midterm oral presentation with eStudio |
| 03/07 | Final design due |
| 03/09 | Midterm presentation |
| 03/10 | Begin implementation and intermittent testing |
| 04/04 | Test plan due |
| 04/06 | Test plan review with instructor |
| 04/13 | Code review with instructor |
| 04/20 | Practice final presentation in Marksbury |
| 04/26 | Finish implementation/testing |
| 04/27 | Final presentation in Marksbury |
| 04/28 | Product delivery meeting with client and instructor |

* 1. Platforms, Tools, and Languages

Our project runs on the Arch Linux 4.4.1-2 operating system, and uses an Apache 2.4.18 server. Our relational database management system is Sqlite 3.11.1, and we developed the frontend using Javascript 1.8.2 and HTML bootstrap 3.3.6. Finally, the framework for our project is Django 1.8.2 (using Python 3.5.1).

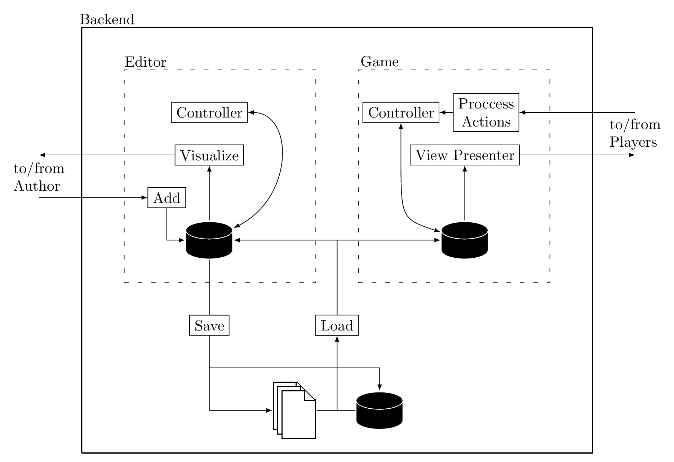
1. Design
   1. Diagrams
      1. Use Case Diagram

This diagram outlines the different users and associated actions that users will be able to make when interfacing with the website.



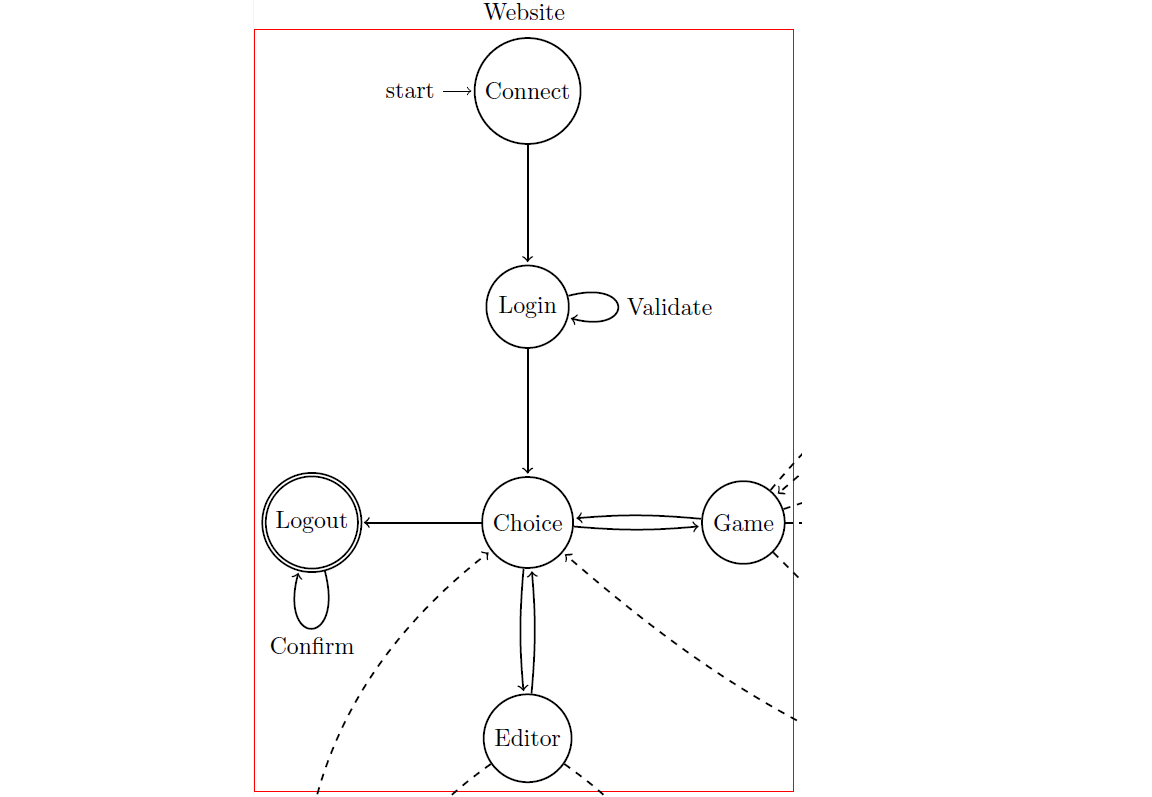
* + 1. Architecture Diagram

This diagram lays out the general architecture of the project. Modularity of design provides flexibility. There is a top level module which implements the landing page for the project and user authentication. The game submodule implements a single instance of a game. This is shared between multiple players. The editor submodule implements a way to create scenarios.



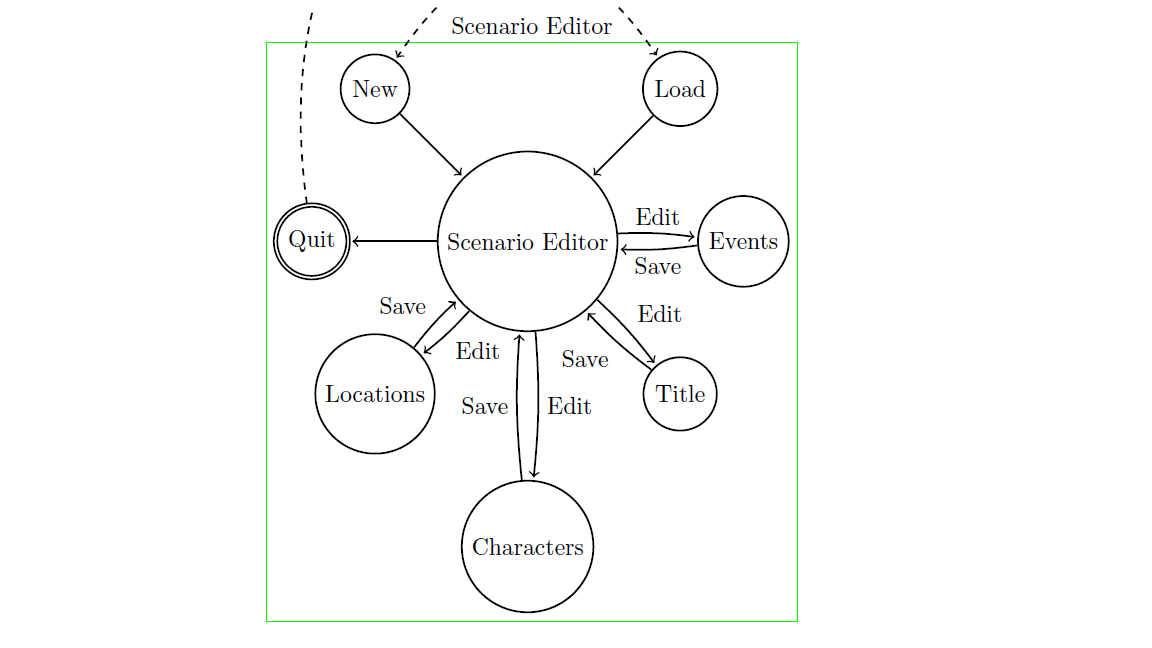
* + 1. State Transition Diagram

This diagram outlines the user’s possible actions over the course of their experience with the project website. The following list includes explanations for each state.



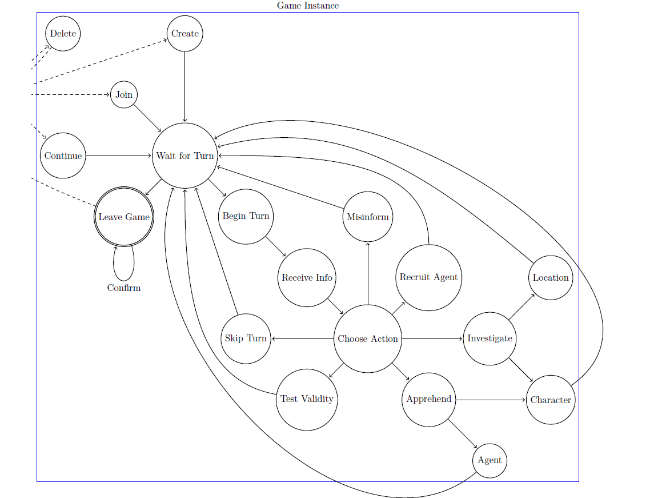
Website State Explanations:

* Connect - starting state
* Login - log in to website, register if not already
* Choice - main screen of site, includes options to logout, enter scenario editor, or manage games
* Game - moves to game manager screen
* Editor - moves to scenario editor screen
* Logout - logs user out of website and returns to login screen



Scenario Editor State Explanations:

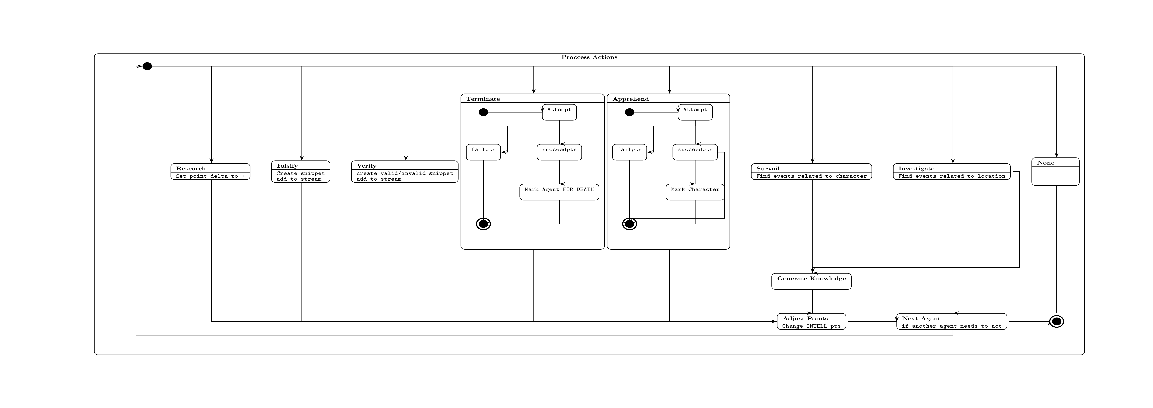
* New - create a new scenario
* Load - load a scenario to edit, must have been created by logged in user
* Scenario Editor - main screen of editor interface
* Title - enter title information (name, number of turns, etc.) for scenario
* Events - add/edit/delete events related to scenario
* Characters - add/edit/delete characters related to scenario
* Locations - add/edit/delete locations related to scenario
* Quit - leave scenario editor interface and return to main screen of site



Game Instance State Explanations:

* Delete - delete a game, must be owned by logged in user
* Create - create a new game instance, logged in user will become owner
* Join - join a new game instance that is owned by another user
* Continue - continue a game instance that is already in progress
* Wait for Turn - wait for next turn to begin
* Begin Turn - indicates that designated time for previous turn has run out and new turn has begun
* Receive Info - user receives snippets of information for the new turn
* Choose Action - user is prompted to choose actions for their agents for the current turn
* Recruit Agent - user recruits a new agent to their team, which increases the number of actions they may perform per turn by one
* Investigate - user may investigate either a character or a location to try and receive more information about the target
* Apprehend - user's agent attempts to apprehend an instance character (an attempt to win the game) or another user's agent (to thwart the other user from progressing)
* Test Validity - tests the validity of a selected snippet so the user may know whether or not the information is pertinent to the scenario
* Skip Turn - this action is selected automatically if the user does not choose an action before the turn timer runs out. This will reward the user with two INTELL points, which may be used to purchase future actions.
  + 1. Action Processing Order

This diagram displays the detailed action processing steps. After each agent’s action is processed, agents are terminated, characters are apprehended, anyone who has captured a key character wins, and snippets are finalized.



* 1. Design Considerations

We realized that there were a number of features less important to core functionality that we likely wouldn’t be able to implement. After discussing our concerns with our client, it was decided that the following considerations would be made during implementation in order to ensure that all requirements were met:

* Randomly Generated Snippets
* Scenario Validation
* Scenario Rating System
* Turn Dependent Win Conditions
* Point and Click Locations
  1. Formal Development Environment

For this project, we decided to follow a loosely defined agile development lifestyle. After beginning implementation, we realized that assigned roles were necessary to increase efficiency. Rather than specify a certain flavor of agile development, like scrum, we decided to minimize emphasis on methodology to focus strictly on implementation within each member’s designated role.

The team decided to set formal meeting times each week on Sunday and Tuesday in order to stay organized and up to date with progress. Communication between members largely existed in person between class times, as well as online through the use of the Slack team communication application.

1. Implementation
   1. Issues

The team ran into a number of issues during implementation, including processing the interactions between user actions, the implementation of isolated databases for each game instance (which the Django framework does not support), and implementing strict stated structure within the framework (which was also not supported). The first two issues were solved through changes to the design itself. The implementation of state was solved by implementing a function that caused the user interface to poll the server for changes, and by creating a chrontab on the server to force the information being sent to the page to update when a user is not currently playing.

* 1. Design Changes

Two design changes were necessary during the course of the project. The first was the inclusion of the action processing order which is described in section 7.a.iv. The second design change was the restructuring of the game instance database so that it did not have to copy portions of the main database into a separate database every time a game is started. This change was made because it was found that the Django framework does not support the aforementioned feature. This was accomplished by folding the game instance database into the main database, so that the issue could be avoided entirely. Incidentally, this also made it easier to create the relations between the various entities within the database.

* 1. Limitations

The team sees no limitations of future work on this project due to implementation choices made at this time.

* 1. Testing

The testplan for this project is available at https://intellproject.com/~dylan/INTELL/documents/testing.shtml. The tests implemented consist of many unit test and limited functional tests.

Some aspects such as logging in were not tested because the Django authorization package was used.

| **Test name** | **Test case (django unittest)** | **Result** |
| --- | --- | --- |
| Root index loads | TopSiteTestCase.test\_index | Pass |
| Login page loads | TopSiteTestCase.test\_login | Pass |
| Login correct | Django package | NA |
| Login incorrect | Django package | NA |
| Register page loads | TopSiteTestCase.test\_register | Pass |
| Register valid user | UserCreationFormTestCase.test\_valid | Pass |
| Register invalid password | UserCreationFormTestCase.test\_invalid | Pass |
| Game page loads | UserCreationFormTestCase.test\_invalid | Pass |
| Editor page loads | EditorViewsTestCase.test\_index | Pass |
| Editor application loads | EditorViewsTestCase.test\_edit | Pass |
| Event model creation | EventTestCase.test\_create\_event | Pass |
| Event with single described by | EventTestCase.test\_single\_describedby | Pass |
| Event with multiple described bys | EventTestCase.test\_multiple\_describedby | Pass |
| Character Creation | CharacterTestCase.test\_create\_character | Pass |
| Location Creation | LocationTestCase.test\_create\_location | Pass |
| Character Creation | DescriptionTestCase.test\_create\_description | Pass |
| Scenario Submission | IntegrationTestCase.test\_sending\_fixture\_with\_title | Pass |
| Character Creation | CharacterTestCase.test\_create\_character | Pass |
| Knowledge Creation | GameTestCase.test\_knowledge\_model | Pass |
| Start next turn | GameTestCase.test\_start\_next\_turn | Pass |
| Games initialized | GameTestCase.test\_games\_init | Pass |
| Add player to pending game | GameTestCase.test\_add\_player | Pass |
| Start game | GameTestCase.test\_start\_game | Pass |
| Create player | PlayerTestCase.test\_create\_player | Pass |
| Process tail action | ProcessActionsTestCase.test\_tail\_action | Pass |
| Process investigate action | ProcessActionsTestCase.test\_investigate\_action | Pass |
| Process check action | ProcessActionsTestCase.test\_check\_action | Pass |
| Process misinform action | ProcessActionTestCase.test\_misinf\_action | Pass |
| Process recruit action | ProcessActionsTestCase.test\_recruit\_action | Pass |
| Process apprehend test case | ProcessActionsTestCase.test\_apprehend\_action | Pass |
| Process research action | ProcessActionsTestCase.test\_reseearch\_action | Pass |
| Process terminate action | ProcessActionsTestCase.test\_terminate\_action | Pass |
| Game index loads | GameListViewsTestCase.test\_index | Pass |
| Game list loads | GameListViewsTestCase.test\_games | Pass |
| Game details load | GameListViewsTestCase.test\_game\_details | Pass |
| Game creation | GameListViewsTestCase.test\_create | Pass |
| Game agent list | GameListViewsTestCase.test\_agents | Pass |
| Game agent details load | GameListViewsTestCase.test\_agent\_details | Pass |
| Join game | GameListViewsTestCase.test\_join | Pass |
| Game player list loads | GameListViewsTestCase.test\_players | Pass |
| Game knowledge list loads | GameListViewsTestCase.test\_knowledges | Pass |
| Game player detail loads | GameListViewsTestCase.test\_player\_details | Pass |
| Game knowledge detail loads | GameListViewsTestCase.test\_knowledge\_detail | Pass |
| Game submit action | GamePlayViewsTestCase.test\_submit\_action | Pass |
| Game play loads | GamePlayViewsTestCase.test\_play | Pass |
| Game get status loads | GamePlayViewsTestCase.test\_get\_status | Pass |
| Game get snippets loads | GamePlayViewsTestCase.test\_get\_snippets | Pass |
| Game get characters loads | GamePlayViewsTestCase.test\_get\_characters | Pass |
| Game get locations loads | GamePlayViewsTestCase.test\_get\_locations | Pass |
| Game get agents loads | GamePlayViewsTestCase.test\_get\_agents | Pass |
| Game get own agents loads | GamePlayViewsTestCase.test\_get\_own\_agents | Pass |
| Game end | GamePlayViewsTestCase.test\_end | Pass |
| Game modals open | ActionsTestCase.test\_actions | Pass |
|  |  |  |
|  |  |  |

1. Future Enhancements/Maintenance
   1. User Interface

Due to time constraints, the team was unable to implement the desired user interface. While the current interface functions properly, it would be nice to “beautify” the product. A themed user interface could enhance gameplay, and would allow for users to have a better experience.

* 1. Move to Larger Server

The current server that the project is hosted from is small and comparatively slow. In order to host many games with multiple players at once, the project would need to be moved to a larger, more robust server.

* 1. Add Scenario Rating System

Since scenarios can be created by users, and gameplay depends largely on the scenario, a scenario rating system would be a helpful feature. This would allow users to rate a scenario after it has been played to completion. Scenarios would be listed with their associated rating, and scenarios with low ratings may have the potential to be removed.

* 1. Add Advanced Character/Location/Snippet Generation

Currently, scenario authors must spend a large amount of time creating characters, locations, and snippets that are not key to the scenario. Every snippet that is shown to players through the course of a game instance must be manually created by the scenario author. This process can be tedious, thus decreasing interest in creating a new scenario. In order to improve the scenario creation process, it would be helpful to implement automated non-key character, location, and snippet generation.

* 1. Test User Friendliness in Depth

Due to time constraints, our testing of gameplay was limited. Since gameplay is dependent on users and the generated scenario, it is impossible to test all possible gameplay situations. It would be helpful to extend gameplay and examine user responses in depth because there may be underlying issues that have not yet been discovered.

1. Conclusions

Over the course of the project, all core requirements set out by the user and in the original requirements documentation were met. Although the core requirements were met, there were a number of design assumptions and project features that changed over the course of implementation. Some of these include:

* Game instances will load and run off of a separate database – This design assumption ended up not being feasible due to use of the Django framework. It was decided to use one database for all game instances instead.
* Incorporation of a themed user interface – The original plan for the project included utilizing exciting user interface for this project, but due to time constraints this was found to be a feature that could not be completed.
* No default game action – A default game action (“Research”) was created in order to allow users the opportunity to earn INTELL points and also not be penalized for failing to log an action in a turn.
* Player action interactions and priority – It was realized that certain user actions might interact with each other during action processing at the end of a turn. To combat this issue, an action processing order was implemented.

Most of the changes to the design were made in order to increase functionality and usability of the application. Aside from the abandoning the implementation of a themed user interface, each change added time and effort to the implementation process. In turn, the schedule ended up getting extended. Extending the schedule meant that there was not as much time to test the user friendliness of the project.

The main difficulty faced during this project, aside from implementing the unforeseen features listed above, was working within the time constraints. The team tried to stay on schedule throughout the semester, but also had to balance the workloads of other classes and responsibilities at times.

Another problem dealt with throughout the semester was dedicating time to learning the languages and framework that were used to implement the project. As mentioned before the schedule was fairly tight, so it was difficult to allocate time for learning these tools. It was realized that spending time to adequately learn these tools was important in order to make the implementation process more fluid.

If the project were to be restarted from the beginning, the team would make a strict and detailed timeline and work to prevent falling behind schedule. The team would also be sure to begin learning the necessary languages and framework sooner in order to save time for implementation. Hopefully enough time would be saved to incorporate some user testing in order to verify that the project is friendly and fun to play.

1. References

"Js-cookie/js-cookie." GitHub. N.p., n.d. Web. Mar.-Apr. 2016. <https://github.com/js-cookie/js-cookie>.

"Almende/vis." GitHub. N.p., n.d. Web. Mar.-Apr. 2016. <https://github.com/almende/vis>.

"Kraiz/django-crontab." GitHub. N.p., n.d. Web. Mar.-Apr. 2016. <https://github.com/kraiz/django-crontab>.

"Django-casper 0.0.4 : Python Package Index." Django-casper 0.0.4 : Python Package Index. N.p., n.d. Web. Mar.-Apr. 2016. <https://pypi.python.org/pypi/django-casper>.

"Django-jenkins 0.18.1 : Python Package Index." Django-jenkins 0.18.1 : Python Package Index. N.p., n.d. Web. Mar.-Apr. 2016. <https://pypi.python.org/pypi/django-jenkins>.

"IEEE Recommended Practice for Software Requirements Specifications." Math.uaa.alaska.edu. N.p., n.d. Web. Feb. 2016. <http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf>.

1. User Manual and Installation Guide

The user manual/installation guide can be viewed as a separate document in installguide.pdf.

* 1. Acquisition

The project can be acquired as a tarball from the following link: https://github.com/dylan-wright/cs499-intell/releases/tag/v1.0

* 1. Requirements
     1. Environment

This project should be installed on a linux machine. While Django can be installed on other operating systems, there is no guarantee that all features will work if not installed on a linux system. This project was originally developed using Arch Linux on the development server but other distributions can be used if other software dependencies are met. This software’s hardware requirements are as follows:

**RAM:** 1 GB minimum. A swap/partition file is recommended, but not required.

**Storage:** Dependent on use. For development purposes, the database required about 0.5 MB maximum. If this software is deployed to many users, this value would grow.

**CPU:** Dependent on use.

* + 1. Dependencies

The following software packages are required by this software. Instructions for their acquisition can be found in section 12.d Acquiring Dependencies.

**Python 3.x**

**Django1.9:** MVC framework used as back end of package. The following additional Django applications are required and can be acquired from the locations specified in section 12.d Acquiring Dependencies.

* **django-crontab**
* **django-jenkins**
* **django-casper**

**SQLite3:** Relational database. MySQL/MariaDB and PostgreSQL can be used with some modifications to the project settings.

**Jenkins:** Continuous integrations server. Dependency can be removed by removing django-jenkins application and JENKINS\_TASKS from settings file.

**pep8/pyflakes:** Optional Python linters which can be used in conjunction with Jenkins build server.

**Bootstrap:** Already installed. HTML pages which require it use CDNs.

**vis.js:** Visualization javascript library. Already installed.

**js**-**cookie:** Javascript cookie library. Already installed.

* 1. Instructions

These instructions are for installation of the product in a development environment. To install the product in a production setting, refer to the Django documentation on the subject at: https://docs.djangoproject.com/en/1.9/howto/deployment/

* + 1. Installation

1. Extract the tarball

$ tar –xvf vx.y.tar.gz

1. Enter the cs499-intell-x.y/ directory
2. Prepare the database

$ ./manage.py migrate

1. Start the server

$ ./manage.py runserver 0.0.0.0:PORT

* + 1. Configuration

At this point, the Django server should be running on the specified port and accessible from any IP address. The next steps are optional but recommended.

* Run the tests (should execute in about 30 seconds)

./manage test

* Create an admin

./manage createsuperuser

* Start the crontab job

./manage crontab add

* Set up a Jenkins CI server (useful but not required)
  + 1. Usage

While running the unit tests from the installation steps should reveal most issues with the installation. To see if you can connect, try to access the root URI of the machine/port the project has been installed on from a browser.

* 1. Acquiring Dependencies
     1. Python 3

On most Linux distributions can be obtained through package manager. Otherwise available at https: //www.python.org/downloads/

* + 1. Django 1.9+

Can be obtained with some Linux package managers (pacman yes, not sure about apt). Can also be obtained using pip. Can also be obtained from https://www.djangoproject.com/download/

* + 1. django-crontab

Installation instructions at: https://github.com/kraiz/django-crontab (uses pip).

* + 1. django-jenkins

Installation instructions at: https://github.com/kmmbvnr/django-jenkins (uses pip)

* + 1. django-casper

Requires phantomjs and casperjs. Casper can be obtained with package manager on Linux or directly at http://casperjs.org/. Installation instructions at: https://github.com/dobarkod/django-casper (uses pip). Must modify /usr/lib/python3.5/site-packages/casper/tests.py to have CasperTestCase subclass StaticLiveServerTestCase instead of LiveServerTestCase (see below):

from django.contrib.staticfiles.testing import StaticLiveServerTestCase #from django.test import LiveServerTestCase

from subprocess import Popen, PIPE

import os

import sys

from django.contrib.staticfiles.handlers import StaticFilesHandler

from django.contrib.staticfiles.views import serve

from django.utils.http import http\_date

from django.conf import settings \_\_all\_\_ = [’CasperTestCase’]

def staticfiles\_handler\_serve(self, request):

import time

resp = serve(request, self.file\_path(request.path), insecure=True)

if resp.status\_code == 200:

resp["Expires"] = http\_date(time.time() + 24 \* 3600)

return resp

#class CasperTestCase(LiveServerTestCase):

class CasperTestCase(StaticLiveServerTestCase):

...

* + 1. SQLite3

Can be obtained with package manager or at https://www.sqlite.org/

* + 1. Jenkins (optional)

Can be obtained with package manager or at https://jenkins.io/. See online documentation for CI server configuration.