

Table of Contents

Research	1
Description	2
Plan of Work Log	4
Project Requirements	5
High-Level Software Design	6
Testing	7
End-User Documentation	8
Team Evaluation and Future Prospects	12
References	13

Research

In order to develop our solution, we looked into what requirements would be necessary, and how our solution would be developed to best fulfill the problem. The problem was related to the high learning curve when it came to learning a first programming language. The most critical component to becoming an effective programmer is the development of algorithmic thinking. We sought to create a solution that will promote problem solving skills and algorithm development in the context of AI development. To do this, we looked at existing “battlecode” (AI) games.

The Google AI Challenge is an annual programming competition in which users develop AI’s to best complete a given task. The task changes year to year, and it accepts a number of programming languages: C++, Java, Perl, Python, Ruby, Haskell, C#, Go, Scheme, Lua, Clojure, Common Lisp. While this programming challenge is advanced and has many of our requirements, it’s too broad and is often overwhelming for beginners, making it difficult for them to participate.

RoboWar is an AI challenge in which you create your own robot AI with the goal to kill other robot AI’s. The programming language used is RoboWar’s own programming language. While this AI challenge is more in tune with what we’re looking for, the use of a programming language that is atypical is not preferred, since the user cannot reuse that language in other contexts.

Battlecode is an AI Challenge run by MIT; it is one of the longest and most popular running programming challenges. It accepts both Java and Scala, and the goal is to write an AI that interacts with its surroundings through a complex API. It has both a Newbie tournament and an advanced tournament, but the game API and complexity is still the same. It also has a set of lectures to teach users how to program with respect to the challenge. This is definitely a useful and enjoyable event; however, the extensive API and complex rules of the challenge make it paralyzing for beginners.

Through our research of “battlecode”-esque challenges, our team was able to recognize the shortcomings of those other challenges and avoid them in our own AI game. We decided to focus on a singular programming language, one that is easy in syntax and dynamic, which will allow the user to focus on the algorithms and problem solving. Additionally, we decided to implement both a simple IDE for the AI development and the simulation interface in the same program. Our game will consist of simple rules and an extremely simple API to allow users to get up and running but also allow users to get as complex as they want.

Description

As software developers, our goal is to create software applications that can be used to solve problems in the real world. One of these problems, which we ourselves have experienced as programmers, is the learning curve for learning how to write algorithms and AI programs. Translating human thoughts and actions to the form of an AI is arguably the most difficult part of learning a language because it requires the programmer to think algorithmically. The development of a programmer's algorithmic ability is imperative to their ability to solve problems, making it crucial in the learning process. Hence, our solution to the problem acts as an IDE (Integrated Development Environment) to promote the development of algorithm creation skills and allows the users to run their AI programs through a game engine.

Our solution takes inspiration from various "battlecode" challenges. Users can write an AI to "battle" against other AI. The software then uses a graphic interface to show how the two AI's act and, ultimately, which one is more strategic and effective in capturing all the nodes. The graphic interface randomly arranges nodes in space, coloring them to denote which AI it represents. The game engine runs each AI simultaneously and depicts the decisions of the AI in the game window. The population of each node is halved each turn (rounding down the amount of units sent), and those units are transferred to other nodes. If the receiving node is of the opposite 'team', that node's population is decreased by the amount of units sent. If the receiving node is of the same team, its population is increased by the amount of units sent. Once a node's population reaches zero, it changes alignment (the side it's on). The nodes change color to show which AI has control over the node, and when all of the nodes are a single color, an AI has won. These simple game mechanics allow for users to quickly grasp the functionality of the game and start programming, but it also allows for more complex strategies.

The expected users are beginning programmers who are learning to think algorithmically, so the software was written in and only accepts python code. Python is a dynamic language, meaning that the interpreter is flexible on object types, allowing the user not to have to worry about the formalities of more classic and static languages like Java or C++. Python syntax resembles pseudo code, making it extremely easy to understand, and also easy to start writing in. The simplicity of python allows the user to push past the syntax and 'formalities' barrier of learning a language and focus on developing their problem solving skills, making it the ideal language for our software.

The software itself comes with four example AI's: pacifist, aggressor, midway, and proliferate. Pacifist doesn't perform any actions and can be thought of as a lack of AI. Aggressor attacks all enemy nodes at once in an inefficient manner. Midway has each node attack the nearest respective node. Proliferate constantly reinforces its ally nodes by healing them. Obviously, these AI's are nowhere near challenging to defeat, and they are more for showcasing the program and how it works.

The main benefit of the software is its ability for the user to develop their own AI's and run them, or those of others, in the game window. While this allows the user to write multiple AI's and run them against each other, this yields the potential for a cooperative learning effort. Users could potentially develop their own AI's and share them with other developers - setting the AI's against each other and learning from that experience. Once the AI's battle against each other, the programmers can analyze the match, exchange ideas on the algorithms they used, and provide each other perspective on problem solving skills and strategy. Problem solving skills are often limited by the individual, so collaboration, which the software promotes, is ideal to the development process.

Hence, with a concise language, a simple UI, straightforward game mechanics, and accessible collaboration, our solution solves the initial problem of a steep learning curve when learning to write algorithms and AI programs.

Project Requirements

- syntax-highlighting
- multi-tab Integrated Development Environment (IDE)
- easy-to-use language for user AI scripts
- built-in AI examples for the user to run and learn from
- run the matches using example/user AI easily from IDE
- easy-to-learn, real-time game-type with potential for strategizing
- up to 7 players per match
- proper documentation for the user to learn the system
- portability, users can use their AI files on any system (that has Python)
- encourage collaboration
- encourage speedy-development of AI code

High Level Software Design

Testing

Random Coordinate Generators

Tested the Random Number Generator using the following script:

```
import random
distro = (1000)*[0]
for i in range(0, 1000000):
    distro[random.randint(50, 590)] += 1
for i in distro:
    print(float(i)/float(590-50+1))
```

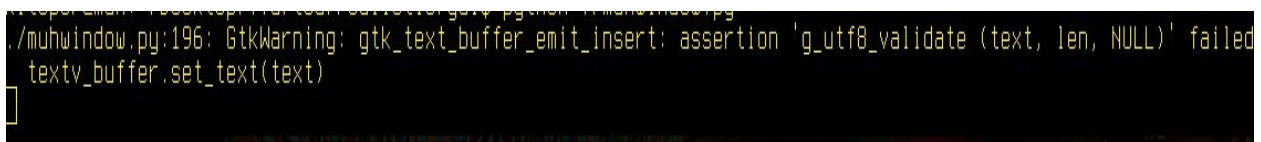


```
3.45656192237
3.24768946396
3.55637707948
3.41035120148
3.43992606285
3.46765249538
3.34011090573
3.53419593346
3.45656192237
3.41035120148
3.36783733826
3.49907578558
3.34011090573
3.46395563771
3.43992606285
3.55637707948
3.32902033272
3.45656192237
3.45471349353
3.54898336414
3.29205175601
3.3752310536
```

shows similar counts, thus accurate pseudo-randomness.

File I/O

Testing for error messages when opening a non-text file.



```
./muhwindow.py:196: GtkWarning: gtk_text_buffer_emit_insert: assertion 'g_utf8_validate(text, len, NULL)' failed
textv_buffer.set_text(text)
```

The text buffer opens blank, as it can't interpret the binary file.

End-User Documentation

The Game

- Start
 - The game can start with up to 7 players: Red, Orange, Yellow, Green, Blue, Indigo, and Violet. Each player is given an equal number of starting nodes.
- Alignment
 - Each node has an alignment. The alignment is determined by the player's color and shows possession. Ex: Red's nodes have a red alignment.
- Population
 - Each node has a population. Every turn, the population of each node is incremented.
- Transfers
 - Transferring is the method of attack and defense for nodes. It's the process in which a node halves its population and sends it to another node via line connection. If the recipient of the sent population is an ally to the sender, the sent population is added to the recipient. The inverse case deducts the sent population from the recipient. If a node's population is reduced to under 0, its alignment is changed to the attacker's.
- Turns
 - Each turn, all players make their actions simultaneously. The players decide which of their respective nodes will initiate transfer with other nodes.
- Goal
 - There is only one winner. The first player to take over all nodes wins.

API Documentation

Class Planet

x - the planet's x coordinate (integer)
y - the planet's y coordinate (integer)
alignment - the planet's alignment (string)
population - the planet's population (integer)

Writing The AI

The player AI consists of a single file. The single file only requires a decision function that accepts an array of planets on the field and the alignment its deciding for. The decision function returns a 2D array of sender-recipient pairs for new transfers to be created. Below are the example AI's included in the program.

Pacifist

The basic format for the decision function. This decision function doesn't do anything with the planets array and alignment, it merely returns an empty array, no new transfers are created.

```
def decision(planets, alignment):  
    return []
```

Aggressor

The aggressor AI attacks, transfers, to all opposing nodes. It loops through the planets array and starts transfers to all nodes not matching the AI's given alignment. Only plants with population greater than 10 attack, because a lower population means less defense.

```
def decision(planets, alignment):  
    transfers = []  
    for i in range(0, len(planets)):  
        if planets[i].population < 10: continue  
        if planets[i].alignment == alignment:  
            for j in range(0, len(planets)):  
                if planets[j].alignment != alignment:  
                    transfers.append([i, j])  
    return transfers
```

Proliferate

The proliferate AI is almost opposite to the aggressor. It loops through the planets array and starts transfers to all nodes that match the AI's given alignment, effectively and consistently healing the team. Like the aggressor, the proliferate AI's planets won't heal unless they have greater than 10 population.

```
def decision(planets, alignment):
    transfers = []
    for i in range(0, len(planets)):
        if planets[i].population < 10: continue
        if planets[i].alignment == alignment:
            for j in range(0, len(planets)):
                if i != j and alignment ==
                    planets[j].alignment:
                        transfers.append([i, j])
    return transfers
```

Midway

Loops through the planets array. For each planet owned by the AI, a transfer is created between the planet and its closest neighbor.

import math

```
def decision(planets, alignment):
    transfers = []
    for i in range(0, len(planets)):
        if planets[i].population < 10: continue
        min_distance = 0
        candidate = -1
        for j in range(0, len(planets)):
            if i != j:
                distance =
                    math.sqrt(pow(planets[i].x-planets[j].x, 2)+
                        pow(planets[i].y-planets[j].y, 2))

                if distance < min_distance or candidate == -1:
                    min_distance = distance
                    candidate = j

        if candidate != -1:
            transfers.append([i, candidate])
```

return transfers

Starting The Game

To start a new game, go to the “Game”. After clicking the “Start New Game” menu-item, you’ll see the assign/remove dialog. By clicking “Assign” you’re given the option of loading an AI from file or an AI example supplied by the application. “Remove” will clear that player’s ai. Any blank player ai aren’t used in that game.

Editing Files

New

Creates a new file tab in the text-editor.

Open File

Opens an existing file in a tab in the text-editor

Close File

Closes the currently-selected file tab in the text-editor

Save/Save as

Saves the currently-selected file tab in the text-editor

Exit

Exits the program

Installation

The application requires Python2.7, PyGtk (for Python2.7), and PyOpengl (for Python2.7). All of these are installed automatically in Windows by the install.bat file.

To find out more about installing and running the program, see README.txt in the root directory.

Team Evaluation and Future Prospects

Andrew Suh

I think that we did a good job in making the basic functionality so that users can actually use it, but I also think that there's potential to add some features to the interface to make it more usable. For example, it would be beneficial to add controls to the simulator window (where the live game is displayed) like speed controls, play/pause, reverse/forward, and replay. Additionally, it would be nice to have a legend to make it clear what AI file corresponds to what color. Lastly, I think it would be useful to add more parameters to the settings menu, such as the amount of nodes per AI, the magnitude of the increase in population per turn, etc.

Christopher Sfalanga

I really liked the simplicity of the end product. Like shown in the example AI's, you can get an "AI" working with just a few lines. In the future, I'd like to create a better IDE for the users. For example, I'd like to implement more specific error checking that displays errors directly in the text editor when the program runs. Moreover, it would be interesting to add different gaming modes with more complex environments and rules, so that users don't get tired of the simple gaming mode. For example, a "collaborative" gaming mode would be interesting, one which the user would run two AI's (working in a team) against two other AI's, adding a whole new level of complexity to the game and offering a new opportunity for collaboration.

Edwin Gomez

I enjoyed playing with the game myself, so in that aspect I definitely think we succeeded. However, I see a lot of potential in the "sociability" aspect that we identified. I envision adding a web presence to this game. My idea is to have a forum or some sort of file-sharing site specifically developed for this game, one where users could share their AI's so that people can simply test their AI's against others worldwide. Additionally, if players do not want to publish the source of their AI's, our web service would offer game matching so that our server would run the simulations with the AI's uploaded by users, and then play back the game to the user.

References

- "Battlecode - AI Programming Competition." *Battlecode - AI Programming Competition*. N.p., n.d. Web. 25 Feb. 2015.
- "Google AI Challenge." *Google AI Challenge*. N.p., n.d. Web. 25 Feb. 2015.
- "Robocode Home." *Robocode Home*. N.p., n.d. Web. 25 Feb. 2015.
- "Windows RoboWar - Finally Here!" *Windows RoboWar - Finally Here!* N.p., n.d. Web. 25 Feb. 2015.

Libraries used for development (all GPL):

PyGtk (pygtk.org)

PyOpenGL (pyopengl.sourceforge.net)

PyGTKCodeBuffer (code.google.com/p/pygtkcodebuffer/)