The A.I. Craft Project

Ray Shulang Lei
aicraft.org
ray.lei@uregina.ca

March 19, 2012

Abstract

Gamify Learning has recently gained popularity and believed to be effective for education purpose. Todays economic has evolved to be "developer-centric"[3]. While we have many games to practice various skills in a fun way, there has not been too much efforts made for development skills.

There is a recently made popular practice called "livecoding", where programmers perform art using their programming skills in real-time to interactively improvise their works on the fly.

A.I. Craft is a web based multi-player artificial intelligence combating game driven by livecoding competition between players. Players provide their code in real-time to control their A.I.s, in order to beat the opposite A.I.s.

1 Introduction

Project A.I. Craft is the development of a livecoding A.I. combating game. According to wikipedia, livecoding can also be referred to as improvised interactive programming. Players use their programming skills to improvise their code in real time to control their A.I.s, in order to beat the opposite players. The idea is inspired by the JAVA Robocode[1] project. The main

purpose is to "gamify" the never-ending learning process for programming. To understand what "gamify" means, here is the definition from wikipedia: "Gamification is the use of game design techniques and mechanics to solve problems and engage audiences."

Not until recently, the dynamic programming languages have envolved to be fast and effecient enough for livecoding practice. With some new technologies I am going to introduce later (HTML5, NodeJS), javascript, is treading to be the unified programming language for the web. Powering by the Google's V8 engine, javascript has been use for livecoding in audio and visual arts at last year's Google IO conference openning show. A.I. craft is aimming to take the livecoding practice further into the video game world, seeing that the current technologies is mature enough.

2 Rationale

2.1 Motivation

In Noah Falstein's article "Natural Funativity"[2], he described three types of hunters in a hunter-gatherer society. After three hunters have got enough haunch of antelope, enough to feed his family for a few days:
Aagh - goes right back out to track down a deer he saw earlier;
Bohg - passes his time by kicking back and catching nothing more challenging than some rays;
Cragh - plays hunting games at home. "Like Aagh, he is building survival skills - but does so it in the safe confines of home. Like Bohg, he stays safe - but also stays fit and slightly improves his chances for success in the next hunt. "

Noah Falstein concluded that "Over hundreds of thousands of generations, those genes that Cragh carries are more likely to spread, and the activities - including games - can also be passed on through word of mouth in the tribe from generation to generation."

According to Venkatesh's article "The Rise of Developeronomics"[3] on Forbes, our society now is a developer-centric economy. He further argues that there are only two kinds of people matters: developers, and people who controls developers. Assuming our society is developer driven, I can conclude that the developers are as important as hunters in a hunter-gatherer society. What Aagh, Bohg and Cragh would be like in today's society? After developers finishes their work before the deadline, there would be three types of behaviors:
Aagh - goes right back out to code for more income, whether it is a contract job or freelancing;
Bohg - will be chilling out, get more sleep, let the brain rest;
Cragh - plays games at home to sharpen his development skills.

Now, let us think about what kinds of games Cragh can use to improve his development skills. First person shooters? It only makes a better hunter. Strategy games and others? Maybe, but they are still not exactly related to coding. Thus we need a new genre of games: coding games.

2.2 Robocode

Robocode is a educational coding game. Players write programs that controls a miniature tank that fights other identically-built (but differently programmed) tanks in an arena. Tanks can move, shoot at each other, scan for each other, and hit the walls (or other robots) if they aren't careful. When I first start looking for coding games, Rococode is the closest to what I am looking for.

Project A.I. craft is inspired by the Robocode, and thus originated from it. Let us look into the designs of Robocode to better understand the A.I. Craft project. Here is a paragraph quote from Robowiki:[4]

"The Anatomy of a Robot is divided to three parts:

The body - Carries the gun with the radar on top. The body is used for moving the robot ahead and back, as well as turning left or right.

The gun - Mounted on the body and is used for firing energy bullets. The gun can turn left or right.
The radar - Mounted on the gun and is used to scan for other robots when moved. The radar can turn left or right. The radar generates onScannedRobot events when robots are detected."

Now let us look at the APIs:
Movement Control:

```
public void run() {
  while (true) {
    ahead(100);
    turnGunRight(360);
    back(100);
    turnGunRight(360);
  }
}
```

Radar and weapon control:

```
public void onScannedRobot(ScannedRobotEvent e) {
  fire(1);
 }
```

These two blocks of the code call the Robot APIs of run(), ahead(), turnGunRight(), back(), onScannedRobot() and fire(). The meanings of these API calls are trivial thus I won't explain them. Notice that onScannedRobot() is a callback funciton when the robot is "aware" of an event - in this case, the radar has discovered an enemy.

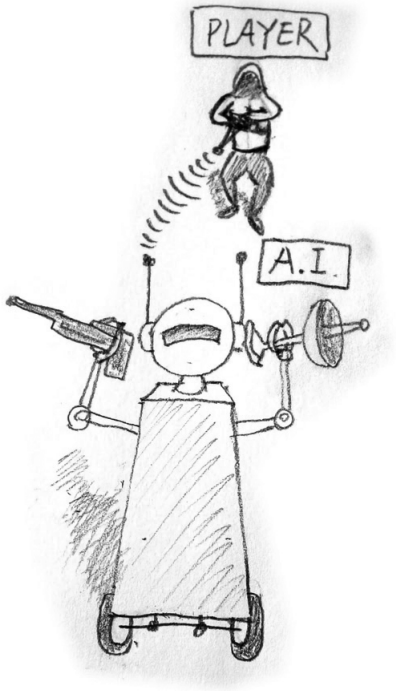
These instructions represents the robot point of view on how to act and react to events. The first code block means "go ahead 100 pixels, turn 360 degrees(thus scanning 360 degrees with the radar at the same time), go back 100 pixels, turn 360 degrees, and repeat this until this robot dies.

Furthermore, the physic engine determines the results of the corresponding Robots actions. Physical engine uses the newton laws to determine the move-

ment/rotation limits. And it also decides how the collisions between robot and bullet, robot and robot, robot and wall will generate damages.

3 The core design of A.I. craft

The basic idea is very similar to Robocode. The term "Robot" in Robocode is now referred to as "A.I.", and similarly, an A.I. has a radar scanner, a mounted weapon and the moving unit.



Players provide A.I. controlling code to tell their A.I.s what to do. The game engine provides 3 sets of controlling APIs such as move, search, aim/fire. The difference between A.I. Craft and Robocode, however, besides being multi-player and online, is mainly about players' ability to program.

To satisfy the goal of gamify learning, letting the players compete with each others on their ability to program is important. That is where the need for livecoding comes in. In the starting of a match, two players with their A.I. in "null" state will be spawned into the game scene.

Players will submit their controlling code from there, thus they will have to take the code that is submitting by the opposite into consideration. For example, an experienced player will probably be submiting attacking code in the first place so that if the novice player is doing the same, they would got the lower hand. Therefore the novice player should consider to submit dodging code first.

Another major different from Robocode is to physically put the players into the arena. One of the purpose is to allow players to submit code to the A.I.s within a limited distance. This will give the players an immersive feeling when playing the game. Another purpose is to provide some new API calls such as follow(), so that the A.I. can follow the player's movement to avoid opposite attacks in the beginning. This "following mode" provides great survivability for novice players even if they are facing an veteran.

4 Technical Details

4.1 NodeJS

"Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices."[5]

In other words, NodeJS is a browser javascript engine running on the server-side for serving network contents, similar to Perl/PHP/Ruby etc.. Since my players will be submitting javascript code, and the javascript code will be run on serverside in real time to control their A.I.s, choosing NodeJS seems obvious. NodeJS will be used to fulfill these features:

1. Serving front end HTML/WebGL files
2. Connecting to database
3. Checking if user submitted code is secure
4. Executing user submitted code in real time
5. Executing and storing game logic in real time

Why NodeJS but not C++/Python/Ruby/LISP ...
Javascript is supported in all major web browsers these days. Having NodeJS allows same code runs both at server side and in browser, and saves huge time in developments.

The current version of the project proves that running user commited code on server side is not the optimal option. In that case, Language/framework/server software of choose is not that relevant anymore. When the user code is running on client side, server only needs to:

1. Simulate the physics engine;
2. motorize the A.I. body when clients send in commands using websockets or whatever.

4.2 WebGL/HTML5

WebGL is a software graphical library that extends the capability of the JavaScript programming language to allow it to generate interactive 3D graphics within any compatible web browser.

HTML is a language for structuring and presenting content for the World Wide Web. At version 5, HTML now provides a rich set of APIs such as socket communication, multi-threading and local storage, to enable rich interactive applications in browser.

Online game clients nowadays have three forms of distribution: Binary executable downloads, running in a browser plugin and running in a browser

natively with no plugin required.

Performance wise, binary executable game clients are superior, since they often are making use of DirectX/OpenGL libraries and pre-compiled at runtime. However, its downside is platform dependent and users need to trust your binary code before they are willing to run it. Not being platform independent will greatly extend the development cycle, thus A.I. Craft will not be using this form of distribution.

Browser plugin games, on the other hand, such as flash games which require a plugin to run in browser, is now considered out-of-dated and unfashion after WebGL/HTML5 have been introduced. Because the trend of more devices will be supporting HTML5/WebGL and less devices will support flash and other web browser plugins, A.I. Craft will be using the third form if distribution: running in browsers natively.

Unfortunately, Microsoft internet exporleror has not provided support for WebGL and Socket communications, therefore IE users will not be supported by A.I. Craft. Firefox and Chrome users will be the targeted audience.

The animation and physical calculation of the game will be running in browser. Features included:

1. Rendering the game scene, player A.I.s and other game objects
2. Calculating collision detections
3. Submitting players codes in real time
4. Preventing players from submitting pre-coded segments

4.3 Physics engine

The physics engine I am using is Bullet Physics. It is open-sourced and used by many commercial projects including PS3 video games. It is written in c++ and has binding for lots of other languages, including a emscripten javascript version.

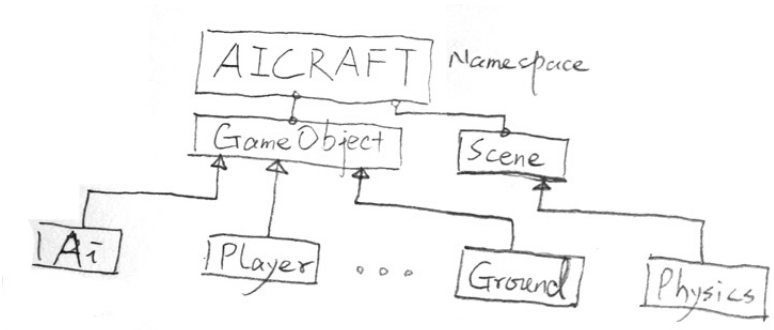
4.4 Software architecture

A very important characteristic of Javascript is that it is event-driven, asynchronously non-blocking. That means if a line of code takes N seconds to finish, the next line of code does not need to wait for N seconds to be started. Instead, it can start executing instantly. This kind of behavior allows multiple events running concurrently. Hence the core of A.I. Craft is three concurrent running engines: Client Engine, Server Engine, A.I. Engine.

"Prototype-based programming is a style of object-oriented programming in which classes are not present, and behavior reuse (known as inheritance in class-based languages) is performed via a process of cloning existing objects that serve as prototypes." – Wikipedia

The whole game of A.I. Craft is based on Javascript, which is a prototype-based programming language. In a classical OOP view, prototyping on an entity means inheritancing that entity.

Here is a very basic layer of the game engine in Class Digram:



The Scene class provides three running loops: Server engine, client engine and AI engine. There are varity of components for physics engine and graphic engine. The GameObject class is very interesting. All classes under GameObject tie to the game loop: they tell the game loop how much they weight, what models they will be using, what is the bounding box/octree they are

binding to for collision dections, and the AI code that controls how they will react to various situtions etc.

4.5 A.I. Coding

The Ai Class is the most interesting one. It is exposed to let players to "prototype" on their A.I.'s intelligence part. Here is a draft idea of what kind of code the players may be submitting:

```
AICRAFT.AI_raysAI.prototype.run() {
  var self = this.body;
  self.ahead(100);
  self.back(20);
  self.turnLeft(20);
}
```

AICRAFT.AI_raysAI is the "Class" extended from AICRAFT.Ai Class. It may seems like the code will let the A.I. go forward 100 units, then back off 20 units, and then turn left 20 degress by its look. In fact, this code is telling the A.I. to perform all these three actions at the same time. The correct way to code the A.I. that the behavior I described is:

```
AICRAFT.AI_raysAI.prototype.run() {
  var self = this.body;
  self.ahead(100, function(){
    self.back(20, function(){ self.turnLeft(20); })
  });
 }
```

You should notice the functional style of the code. This is due to the non-blocking nature of Javascript.

Each code submitting would only allow one prototype in one function, thus forcing the player to "livecode" their A.I., but not copying a big chunk of existed code which is equal to cheating.

By prototyping on the A.I. object, players therefore intractively improvise their A.I.s on the fly. And this is the core concept of the A.I. Craft project.

One thing I need to decide is that where and how I should run my user submitted A.I. code. After some research, I found that I have three options:
1. Having the code run inside the server game engine.
2. Having the code run on server within another jailed process.
3. Having the code run on client side.

It's obvious option 2 and 3 is secure by nature, while being more time consuming compare to option 1. I have chosen to do option 1 due to the experimental purpose of this project.

There are two ways that I can intercept insecure code(that means the code that crashes the server of have unexpected side-effects:
First is the use regular expressions to take out the dangerous code during its string form.
Second is to use try ... catch ... block to handle exceptions.

However, using these two tools to make the user submitted code secure are almost impossible. So in public version, I will be using option 2 or 3.

4.6 network

Network communication is using WebSockets protocol. Which is on top of TCP/IP. While it's not as fast as UDP, it does garantee the packet orders and handles the connection/disconnections nicely. A.I. Craft engines compressed all the network packets to as minimum as possible. Keybord inputs are compressed to bits using bit operators, and the final packet sending to server from clients is just one byte. Same goes for position syncing packets. No text information is transmitted on network.

Another nice feature provided by Node.JS and Javascript is the asynchronouslism. Network transmition can be completely free from main game loops, thus

keyboard byte, position bytes transmition can be done 20 times per second, while the game loop is running at 60 times per second.

Futher more, between two network packets, there is 2 frames of missing detail. $20/60 = 3$, 3 times more detail, get it? Now, since Physics engine running both on server and clients, therefore the 2 frames missing detail after every network syncing frame can be calculated and filled. The physics simulation is reasonably deterministic given the same initial state and inputs. That means if I tell the clients the initial velocity and position for game objects, it will calculated the the "good enough" game state for next couple frames. These approach allows me to have the network syncing runs at 20 frame per second, and the users still looking at nicely 60 frame per second animation on their screen. Best part of all, since the game loop, network loop are seperated, the code to implement this technique is very natual.

During the implementation, it turns out that having the physical engine running inside browser is either causing severe low game performance or constantly crashing conditions. Furthermore, the natural of javascript running inside browser reveals most of the source code running on the server side too. This will pose serious security threats. Thus in the current version, physical engine on client side is disabled.

As the current state of the game shows, it is possible to have the game client and server communicating at as high as 60 frame per second using websockets. However, the delay is pretty bad. Luckly, A.I. Craft does not demand high ability to real-time issues by design. Furthermore, WebRTC which is UDP base client communication protoco is supposed to be polished soon enough, so that an ungrade of the network part is possible in foreseeable future.

4.7 Quaternion

Instead of using rotation matrix for all the turnning, aiming, spinning etc., I use quaterion to represent all the rotation/orientation related information.

The sync the rotation/orientation across loops and network using axis-angle or matrix is a huge hassle. While matrix being too big in data: 9x9 in size, axis-angle requires too much convertions. Quaternions solves my problem easily. A vector4 is all I need to rotate or point anything to anywhere.

4.8 Animation

Animation is produced in Blender with FK skeleton between two to five key frames. And then the outputed animation will be 30 frames of animations, constructed by 30 arrays of vertices. the arrays of vertices then are loaded into the game engine as 30 geometries, but the render output is 60 frame per second, where the interpolated frame is calculated using morph algorithm.

5 Game Instructions

5.1 Installation

First of all, to run this game, you will need a WebGL supported Browser with a WebGL enabled video cards. Recommand and tested browser is Google Chrome `https://www.google.com/chrome`. Firefox, Safari and Opera works.

To run the game server, you will be internet connection (not behind a NAT). Supported Operating Systems are Linux and Mac OSX.

To run the server, you will need Node.JS `http://www.nodejs.org`.

To start the server, simply run the command "node aicraft_server.js"

To compile the source code, run the command "./cmake.sh", java is required. You don't need to compile the code since javascript runs universally anywhere.

5.2 How to play

After you enter "[server name/ip address]:3003/game" in the browser, a prompt pop up will ask you the name of your A.I. Enter the name, and you will see yourself in a game world with a rat like robot besides you. To move around, use these keys:
w - move forward
s - move backward
a - move left
d - move right
q - turns left
e - turns right
alt - enables code emitter

So you see that your robot is not moving at all. Now, after hit alt once, you see a editor pop up with following lines:

```
AICRAFT.ai_name_to_replace.prototype.run = function() {
};
```

The run function is where we move the robot by using these set of calls:

```
this.body.ahead(units, callback_funciton);
this.body.back(units, callback_funciton);
this.body.turnLeft(degree, callback_funciton);
this.body.turnRight(degree, callback_funciton);
```

The function of these calls are obvious so that I will not explainning then. Another set of calls are sight control functions:

```
this.body.lookLeft(degree, callback_funciton);
this.body.lookRight(degree, callback_funciton);
this.body.lookAt(degree, callback_funciton);
```

This one is weapon control functions where it is called in onSightFound function. onSightFound function is called whenever an enemy is spotted in sight

of your robot:

```
AICRAFT.ai_name_to_replace.prototype.onSightFound = function(event) {
this.body.fireAt(position, callback_funciton);
};
```

Once your robot fires at something and hit it, the server console will report the event. Further animation to represent these are being implemented.

5.3 API documentation

A detailed API documentation is generated by JS Doc. Under doc folder, open index.html will gives you a full description for all the function calls.

References

[1] Mathew Nelson, Flemming Larsen, IBM etc.
Robocode.
`http://robocode.sourceforge.net`
2001

[2] Noah Falstein,
Natural Funativity.
`http://www.gamasutra.com/view/feature/2160/natural_funativity.php`
2004.

[3] Venkatesh Rao,
The Rise of Developeronomics.
`http://www.forbes.com/sites/venkateshrao/2011/12/05/the-rise-of-developeronomics/`
2011.

[4] *Robot Anatomy.*
http://robowiki.net/wiki/Robocode/Robot_Anatomy
2008.

[5] Ryan Dahl,
NodeJS official site.
http://www.nodejs.org

[6] Ari Patrick,
Why NoSQL databases are better for games.
http://gamedev.stackexchange.com/questions/5316/nosql-is-it-a-valid-option-for-web-based-game

[7] Michael Kennedy,
MongoDB vs. SQL Server 2008 Performance Showdown.
http://www.michaelckennedy.net/blog/2010/04/29/MongoDBVsSQLServer2008PerformanceShowdown.aspx
2010