The A.I. Crafter Project

Ray Shulang Lei
aicrafter.com
ray.lei@uregina.ca

February 2, 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. Crafter 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. Crafter 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. crafter 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 wait a second and think about what kind 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. If Noah and Venkatesh are right, developers who plays coding games are more likely to spread their genes. And right here I have found the goal of making A.I. Crafter: to imporve the chance of spreading my genes. Therefore it makes perfect sense to put my best effort fulfilling it.

2.2 Robocode

Project A.I. crafter is inspired by the Robocode, and is originated from it. Let us look into the designs of Robocode to better understand the A.I. Crafter 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 movement/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 of A.I. crafter

The basic idea is very similar to Robocode. The term “Robot” in Robocode is now referred to as “A.I.”.

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. Crafter and Robocode, however, besides being multiplayer 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 what kind of code the oppsite is submitting into consider. 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 probably got shot. Therefore the novice player should consider to submit dodging code first.

The reason of physically putting the players into the arena is to provide a new API call - 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. The follow() API call is a perfect example of “dodging code”.

4 Technical Details(Draft)

4.1 NodeJS

The server side will be back by NodeJS. Since users will be submitting javascript code 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

4.2 WebGL/Javascript in Browser

The client side part of A.I. Crafter will be back by Html5/WebGL in your browser. The animation and physical calculation of the game will be in this part. 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 Database engine

The databases will be seperated as client and server sides. SQL is my first choice due to pass experience. NoSQL might be considered with further research. Basic idea is that game objects information stores at client side databases, and they sync with server side database once a while in a timely basis according to network performance.

5 Public Launch

I am curretnly aiming to launch the preview version of this game online at www.aicrafter.com. The set release date is April 15th.

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