WATonomous Coding Challenge

simulation

This challenge is due on January 9th, 2018 at 11:59 PM

Overview:

The simulation subteam is currently looking for candidates seeking a Software Engineer type role where you'll be working with a large codebase to build a full-fledged product.

The purpose of this challenge is to provide you with an open-ended challenge that you can make as complicated or simple as you want/have time for. Take this as an opportunity to highlight your skills as a programmer. Writing clean, readable, documented and well-structured efficient code is much more important than having lots of poorly written features.

You can use any language you are comfortable with, however it is strongly encouraged and recommended that you use C++.

Requirements:

The challenge is to create a turn based bot fighting game between 2 bots.

The game is played on a N*N (~10 <= N <= ~30) board where the objective is to move around the board while collecting coins and items until either one bot is eliminated (health becomes zero) or after 1 minute of play. If time runs out, the bot with the most points wins.

Bots can move up, down, left, right, but not diagonally.

Each tile on the board can contain an item or nothing. There should be 4 different possible items with 4 different attributes:

1. increase collecting bot's health
2. increase collecting bot's points
3. decrease enemy health
4. decrease enemy points.

Don't worry about implementing other types of items. Each tile can contain at most an item or bot.

The size of the game board should be passed as a command line argument to the program. The game board is initialized such that each bot starts at a random square and has at least 5 squares between it and the other player.

Some number of items are randomly placed on the board. Please pick sensible thresholds for the number of each item placed on the board. The board should not be completely empty but should also not be completely full of items.

Once the board is initialized, each player receives a random starting position. Every turn, each bot decides which direction to move. Once both bots have decided on a move, they make their moves simultaneously.

If both bots end up moving to the same square, a crash occurs and both bots are destroyed and the game is over. The bot with the most points wins in this case.

In a player vs player match, the game must wait for each player to select a move. In a player vs computer match, the game only needs to wait for the human player to select a move.

The computer should make its decisions without knowing the human player's choices (please don't make a cheating computer). For computer vs computer match, wait for input from the user to advance the game turn by turn.

Human player controls should be sensible. I recommend arrow keys and or WASD.

Levels of Implementation:

As stated earlier exactly what you implement and how you implement it is completely up to you. Here are some examples of what's expected at various levels of implementation.

Please don't feel pressured to implement the highest level, just focus on creating the best implementation that you can do with the highest code quality possible. Below is a rough guideline of the levels of complexity that you may strive for:

1. Simple Text: Implement a player vs player game using the text console to receive input and output the text representation of the board after each turn.

2. Basic Text: Extend the above to support the other 2 modes of play (player vs computer and computer vs computer). Use command line arguments or ask the user before the game starts which mode of play they want. Use abbreviations PvP (player vs player), PvC (player vs computer) and CvC (computer vs computer) to represent the game mode. Computers can use any strategy you want, choosing a random direction is fine.

3. Add Graphics: Add a graphical user interface to the game using a platform-independent GUI library (I recommend Qt, but feel free to use any other if you are more comfortable, mention it in the README). The design of the interface is completely up to you. Add whatever you feel is necessary to make a user friendly and informative display.

4. (Bonus) Network Multiplayer: At this point you have an awesome bot fighting game but it is restricted to being played on one computer, let's fix this. Split your one executable into 3 parts: the server, player1, player2. When the server starts up, a game mode should be selected. Once selected it will wait for 2 players (computer or human) to connect. Once the players are connected to the server, it should facilitate the game by accepting moves from both players, advancing the game a turn, and informing the players of the new board/game state. This communication should be done via TCP/IP. Players should accept via command line whether they are human or computer. Players should just use the "Basic Text" output mechanism. Only the server needs to display the GUI. When testing, all three executable should be run on the same machine by using localhost:portNum to distinguish them. Keep in mind that this is a bonus, it is not expected that you attempt this. However, because of its difficulty, candidates that successfully well engineer this feature will stand out from the crowd. Documentation for this feature must be submitted for full credit to be recognized. Please document and explain your protocol and why you did what you did, we look forward to reading it (but also don't make it 10 pages).

What to submit:

All your code and documentation should be placed in a folder called BotFight-<firstname>-<lastname>, compressed and emailed back to your interviewer.

In addition, add a README file to the folder, It should contain an explanation of what you implemented and any additional features you added. Please keep this brief and to the point, with so many applicants we don't have the ability to read 10 page manuals.

Also include compilation instruction so we know how to build your application, and any additional dependencies/packages it depends on. Please include an explanation of any command line arguments you have.

To make your interviewer the happiest, all they should have to do to run your application is:

1. cd BotFight-<firstname>-<lastname>
2. make
3. ./botfight <command line args>

It's also a good idea to include a short ~10 second recording of your application in action. Use something like obs to capture the screen recording.

IMPORTANT: Partial work is better than no work at all- submit a write up containing what you were able to complete and why you weren't complete the challenge in time and/or what roadblocks you hit.

Feel free to send questions to jbmalchi@uwaterloo.ca if have any questions.