

# CODE CAMP CHALLENGE v5.0

## BELL TOWER ESCAPE!

### Background

Soon after Jurassic Park re-opened, some of the dinosaurs were purchased by the Pillsbury family for a Minneapolis branch of Jurassic Park. Unfortunately, they escaped and have made their way to the GMI campus. During an East Wing wide meeting on every floor of the Bell Tower involving network latency, the dinosaurs started getting in...! Due to the inciting panic, and velociraptors taking over the lower levels of the stair well (the doors only open inwards - so there's nothing to worry about!), the Bell Tower Elevators have been overloaded with people trying to get out of the building and away from the velociraptors!

Each bank is controlled by a different micro-controller, and elevator engineers have asked you to program one of them, and another colleague to program the other. They figured one of your algorithms will be efficient enough to get everyone where they want on time!

### Game Information

You get to control two elevators that go up and down the twelve floors. You know what floor people are on and find out what floor they are trying to get to once they enter your elevator. Each turn consists of either moving an elevator up a floor, down a floor, or allowing people to get on or off (people will get on and off in the same turn). If an elevator from both banks opens on the same floor, the group will go for the elevator with fewer people, until the elevator's capacity is met, then people will go into a different elevator.

The game is turn based with 500 turns and the person who has 'delivered' the most amount of people in that time is declared the winner. When people get on your elevator, they will have a 'patience' value that starts at

$$abs(destFloor - currentFloor) * frustrationCoefficient + 3$$

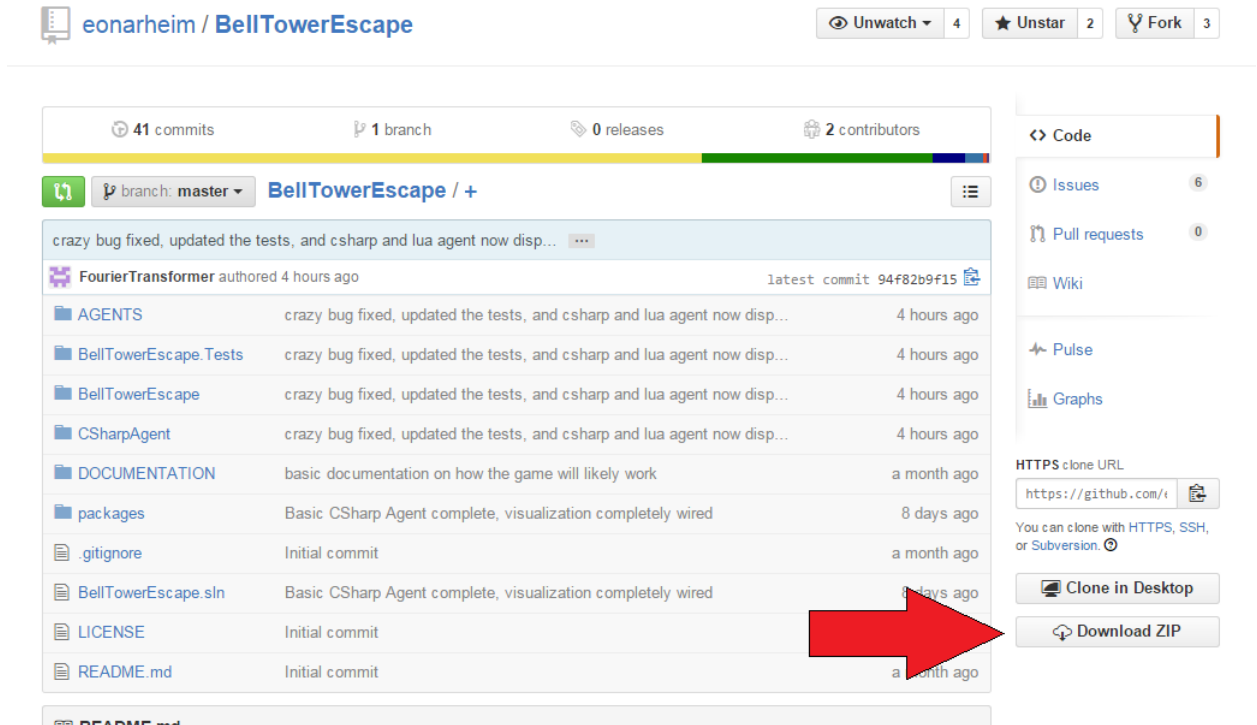
where  $frustrationCoefficient = 2$ . This value decreases by 1 every turn and if it's negative the Meeple will get off at the next elevator stop and count negatively towards the number of people delivered.

### THE COMPETITION

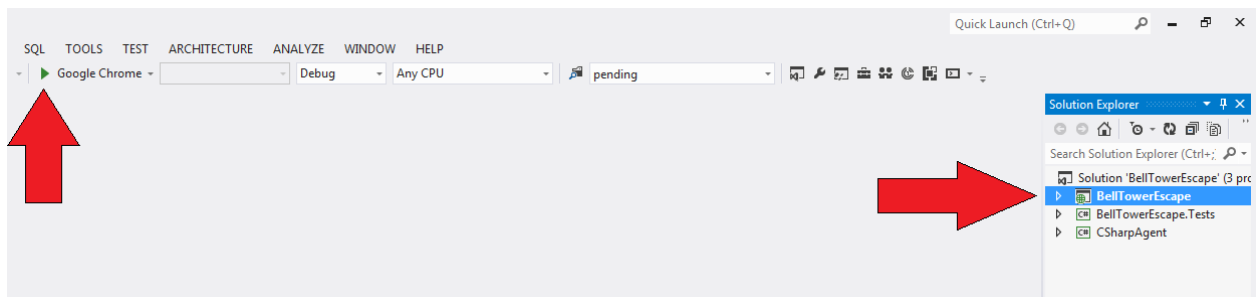
Once we've gathered all the submissions we'll pit them against one another and determine who has the best AI of them all!

## Setting it up locally

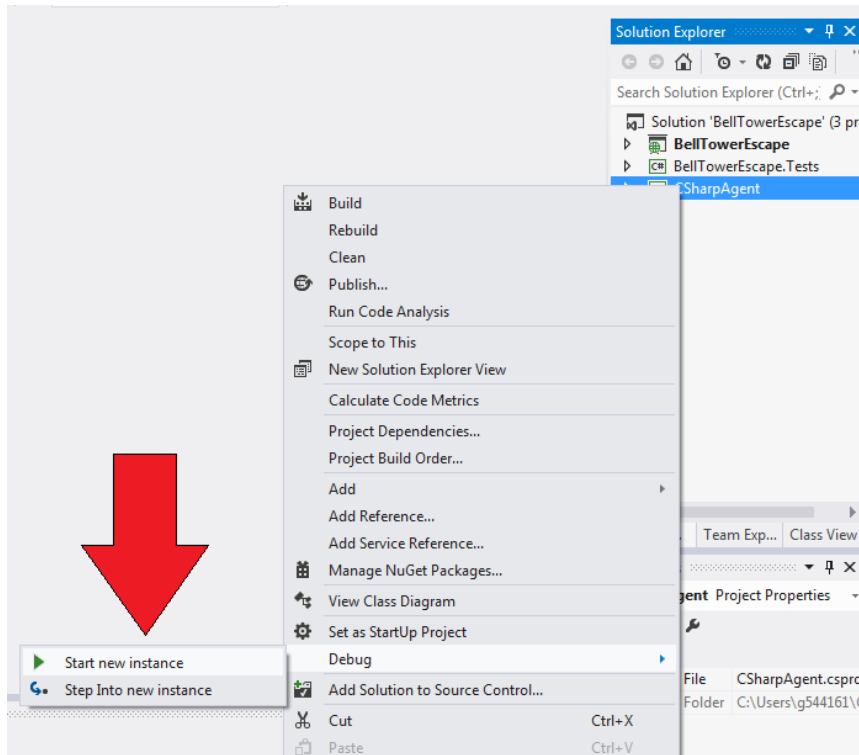
1. If you're familiar with git: `git clone http://github.com/eonarheim/BellTowerEscape` otherwise head over to <http://github.com/eonarheim/BellTowerEscape> and click on 'Download Zip' from the right side of the menu.



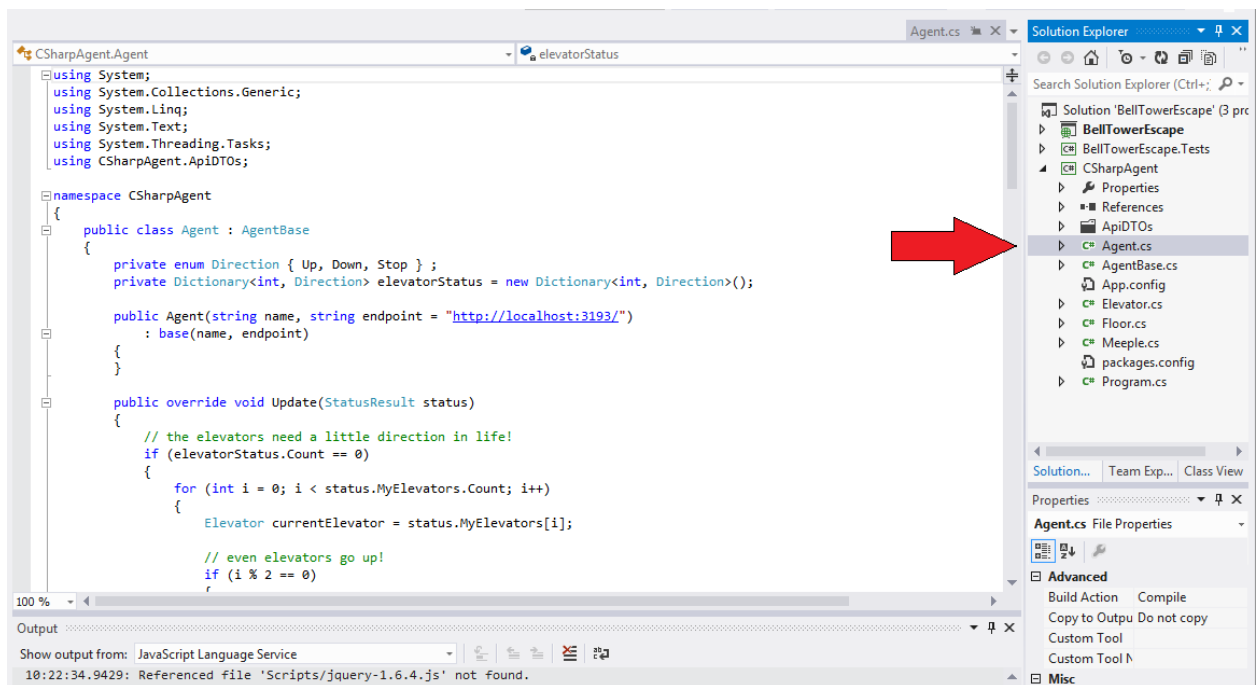
2. Open up `BellTowerEscape.sln` in Visual Studio. If you don't have an MSDN license, you should be able to use the Community Edition of Visual Studio (<https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx>) to run the code.
3. Visual Studio will likely want to get some packages or update some things. Just kind of let it do its thing. If you need some help, reach out to someone in App Dev and they'll likely be more than happy to help out!
4. Click on 'BellTowerEscape' in the Solution Explorer and the little Play button to build it and use the little drop-down to launch it in your favorite browser. It might start doing a 'Nuget Package Restore'. Again, just let Visual Studio do its thing.



5. Once it's loaded up and you can see the web page built, you can start up an agent! The easiest one to get going is the CSharp agent. All you have to do is Right-click 'CSharpAgent' in the solution explorer, click on 'Debug' and then 'Start new instance' and that should get it going!

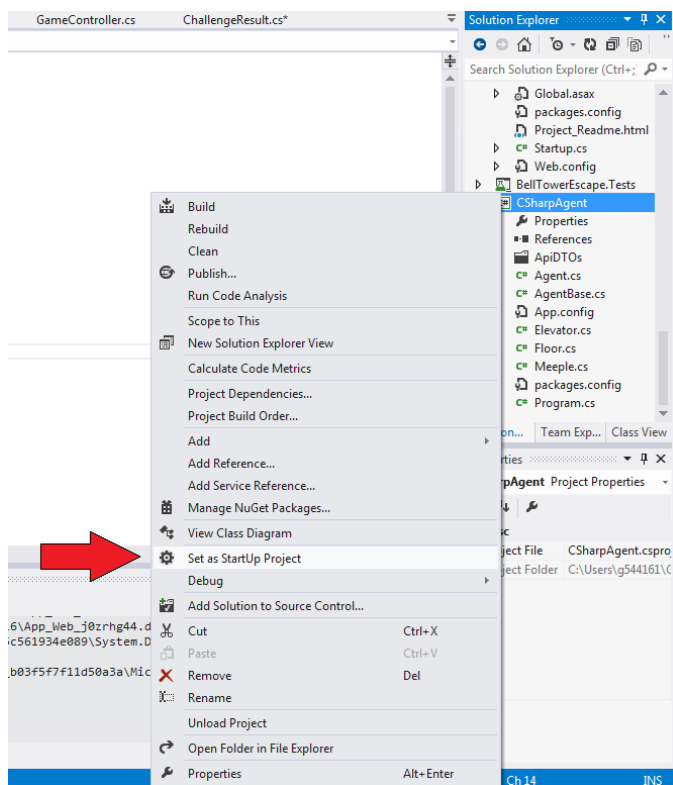


6. SWEET! Now that you got it running, I'm sure you want to make some changes! Click the little drop-down next to 'CSharpAgent' and open up 'Agent.cs'. This file should be all you need to modify to create your own AI. To change the endpoint or agent name look into 'Program.cs'



## Setting it up against Azure

For people who just want to jump right in and not worry about server stuff, we also have BellTowerEscape running in the cloud! The key thing here is to change the endpoint location from `http://localhost:3193/` to `http://elevators.azurewebsites.net/`. This can be done in 'Program.cs' for the CSharp agent, at the bottom of 'PLAY.lua' for the Lua version, and near the bottom of 'SampleAgent.py' for the Python agent. Also, for the CSharp agent, it'll likely be beneficial to make the CSharpAgent the main project by right-clicking and choosing 'Set as Startup Project', so when you build the solution, it won't try to build and run the server code!



## Pitting two AI's against each other

By default, whenever you connect to the game, it'll start playing against the demo agent that uses the Elevator Sweep Alogrithm (described in detail below). However, if you want to pit one of your AIs against another one - without having to modify the server code - it can be done! If you change the logon command from `/api/game/logon` to `/api/game/logonPlayer1`, the server will wait up to a minute for the `/api/game/logonPlayer2` command to be executed and then the game will start (if it never receives it, it'll kill the game). `logonPlayer2` requires you send in the `authToken` and `GameId` that was returned from `logonPlayer1` to start the game. If you need any help with this feel free to reach out to Erik or Shakil.

## Visuals

You can also watch your game happen in realtime! If you are running locally, a web browser window should open up when you click the little play button and you can click on your GameId on the main screen to watch your elevators go! If you're running it against Azure, just head over to `http://elevators.azurewebsites.net` and click on your GameId there!

## The Elevator Sweep Algorithm

This is the current algorithm that is used in all the sample agents as well as the default one that runs on the server. Each elevator is set to move up or down. As it goes in that direction, if there are other people to pick up who also want to go in that direction, those people are picked up! It then continues dropping people off until it reaches the highest/lowest floor, switches direction, and then continues. This is similar to how non-smart elevators work today.

## The API

The API can be found here: <http://elevators.azurewebsites.net/Help>. However, if you're using one of our Agents as a base, you likely don't need to worry about it and if you're creating your own agent, it might be a little easier to base it off a pre-existing one!

## Troubleshooting Steps

If your Package Restore is disabled, the following steps should help you build the solution:

1. Tools → Nuget Package Manager → Manage packages for solution → Updates → click the Update All button
2. Restart Visual Studio
3. In package manager Console execute: 'Update-Package reinstall Newtonsoft.Json'
4. Rebuild Solution
5. Try running it!

## Issues

For any issues with the code, please create an issue on our GitHub page (<https://github.com/eonarheim/BellTowerEscape>). If you have any questions feel free to reach out to Erik Onarheim or Shakil Thakur, as they will rather enthusiastically discuss anything.

## Submission

To submit your AI, place a folder with your name on it in `\\genmills.com\corporate\PUBLIC\codecampv5` by midnight on Monday July 20th. You are allowed to program in any language that you choose (we have provided a Python, CSharp, and Lua agent). We ask that you submit a working Windows binary with the endpoint set to `http://localhost:3193/`, along with your source code and any special build/run instructions (especially if you created a custom agent).