Lab – Connect 4

# Summary

This lab consists of writing a Connect 4 game. The rules of the game can be found at <http://www.wikihow.com/Play-Connect-4>

Please read the entire lab before starting any work.

# Team size

4 people

# Deliverables

1. A minimum viable product, consisting of the following:
   1. Support two human players
   2. Ask a player for a move and parse the user input
   3. Perform the move (adding a piece to the board)
   4. Draw the board, with pieces on it
      1. Need to differentiate between the piece owners
2. Binaries built and released from VSTS
   1. I should be able to download and run your game from the release fileshare. If it’s not there, it’s not done.
3. A sprint retrospective, shared with the class
4. Once the MVP is done, a working connect-4 game (as time allows)

# Requirements

## Form a team

1. Form a team based on the lab team size
   1. Try to work with different people than you have before
2. Create a team name
3. Send me your team name and all team member aliases using Skype
   1. I will create a team and area path for your team in VSTS
   2. The Area Path for your work items should be Cohort#\Labs\<LabName>\<YourTeamName>

## Perform sprint planning

Don’t rush through this, it should probably take 30-60 minutes.

1. Think about what user stories are required for the game to work. Use VSTS to create a work item for each user story
2. Break down the user stories into smaller tasks. Use VSTS to create work items for each task, using child tasks as appropriate.
   1. **You should be discussing business requirement tasks, not implementation details.**
   2. The Area should be Cohort#\Labs\<LabName>\<YourTeamName>
   3. The Iteration should initially be Cohort#\Backlog.
   4. This Area/Iteration combination is your product backlog.
   5. Fill out rough time estimates for each task as a team.
3. Write a query that gets a tree view of the user stories in your team’s area, along with all their child tasks, sorted by priority ascending.
   1. Save this query as a shared query so anyone on your team can run it.
4. Once the query works, prioritize the user stories, and tasks within the user stories.
   1. Remember, Agile is about delivering something that runs (even if it’s incomplete), then iterating on it.
5. Find the tasks that make up your minimum viable product (MVP), and change the iteration for those tasks to Cohort#\Iteration1.
   1. This is your sprint backlog.
6. Distribute the tasks amongst your team.
7. Each engineer should create child tasks of their business tasks which include implementation details (data structures, algorithms, etc…) and technical requirements
   1. The child tasks should each have a time estimate. The time estimates of child tasks should not be more than the parent task. If they are, discuss it with your team.

## Create team branch

1. Browse to <https://microsoftleap.visualstudio.com/_projects>. Click on the project for your cohort.
2. Click on Code -> Branches in the top ribbon
3. Select the repository for this lab
4. One person should create a new remote branch from master that matches your team name. This will be your team branch

## Create build automation

1. Click on Build and Release -> Builds in the top ribbon
2. One person should create a build definition for your team branch. Name the build <LabName> - <yourTeamName>
   1. You can just clone the build called “<LabName> - master” and make the appropriate changes (name, code source, etc…)
   2. Kick off the build manually to make sure it works before moving forward.

## Create release automation

1. Click on Build and Release -> Releases
2. One person should create a release definition for your team branch. Name the release <LabName> - <yourTeamName>
   1. You can just clone the release called “<LabName> - master” and make the appropriate changes (name, artifact source, environment variables, etc…)
   2. This release should automatically be created when a build completes, and the Dev environment should automatically deploy when the release is created.
   3. Kick off the build manually which will trigger the release when it completes. Make sure the Dev deployment completes successfully.
3. To see the released files, open a command prompt and run the following command
4. net use Z: \\leap.file.core.windows.net\test /u:AZURE\leap Y9m0DQpWqoAwxUWRFhsNxwieR4D9p/gUEX6sWN/E77rRw/INBkgcTlIwZDXCbVbR/WNyc1Y0ytUZXgVGfF02+g==
5. Browse to your released files
6. Open windows explorer and browse to the Z drive
7. Double click on the name of your release definition.
8. Double click on the name of your environment (Dev, in this case).
9. Double click on the release number (Probably Release-1, in this case).
10. Your build artifacts should be in this folder

## Create personal branches

1. In VSTS, each team member should create their own remote personal branch from the team branch. Name it something like <yourAlias>-dev
2. Each team member should clone the remote repo, which will get you a local master branch.
3. Each team member should create a new local branch from their personal remote branch. Give it the same name as your personal remote branch.

## Execute the sprint

1. Each team member should commit work into their local branch and push it to their personal remote branch.
2. When ready, each team member should create a pull request from their personal remote branch into the team branch.
3. The team should perform code reviews to make sure the team branch stays healthy.
4. The team branch should always build and release properly. If it doesn’t, it should be fixed immediately.
5. After the MVP is finished, take the next highest priority item from the product backlog and work on it. Continue to do this until you run out of time.

# Code requirements

1. All code should be contained in classes.
   1. The main method should only instantiate the game class and start the game with game.Start() or something similar.
2. All classes and functions should be small and reusable whenever possible.
   1. Each class should have a single responsibility
   2. Each function should do one thing
3. All code should be very well commented. At a bare minimum:
   1. Each class should have XML style comments describing what it does.
   2. Each function should have XML style comments describing what the function does, the inputs, and the outputs
4. All functions should include input validation
   1. If inputs are invalid, the appropriate exception should be thrown
5. **All functions (except the game loop) must have accompanying unit tests.**
   1. Test normal cases, edge cases, and exceptions (if you use them)!

# Tips

1. I’ve provided a significant skeleton for this lab, which should get you up and running quickly. I’ve done this so you can focus on writing the game the logic. Future labs will have more object-oriented design tasks. Feel free to modify the skeleton as you see fit.
   1. All the comments describe the work that needs to be done
   2. Anywhere you see “throw new NotImplementedException()” is just there so the code compiles. You need to remove this after you add in the correct logic.
2. The most time intensive task in this lab is checking for four in a row.
3. It might be a good idea to create private helper functions in the board class to check for horizontal, vertical, or diagonal four in a row, and split these tasks up so they can be worked on in parallel.
4. Make sure all win conditions are unit tested.
   1. For example, for horizontal, the dropped piece can be the first, second, third, or fourth piece. So there should be four tests for just horizontal.
   2. For vertical, the dropped piece can only be the top piece (since all pieces enter the board from the top), so this is the easiest check. There’s only one test required.
   3. For both diagonals, the dropped piece can be first, second, third, or fourth. So there should be four tests for each diagonal.

# Stretch goals

1. What if we wanted to add more players?
   1. Can you write your code such that the number of players is arbitrary and can be set in the config file?
2. What if we wanted to change the board size?
   1. Can you write your code such that the number of rows and columns are arbitrary and can be set in the config file?
3. What if we wanted to change the “win state” to three-in-a-row or five-in-a-row?
   1. Can you write your code such that the number of checkers in a row is arbitrary and can be set in the config file?