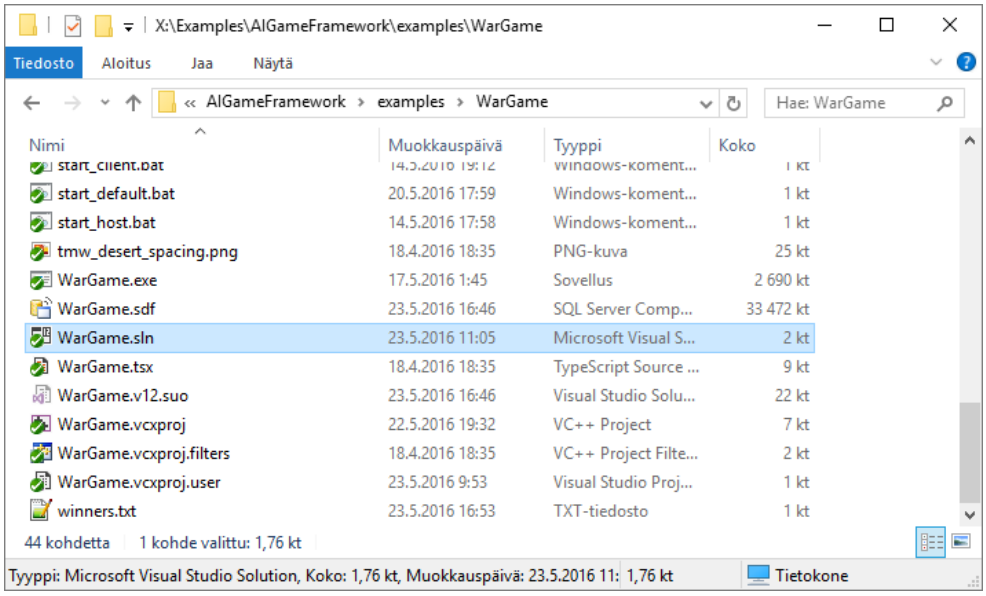
Before you begin

You AI is going to be linked to main "tournament executable" as a static library. Static library must contain every piece of your AI code. Here are instructions for person named "John Doe". He's name will be used as an example name. When following these instructions, please use your name instead. ;)

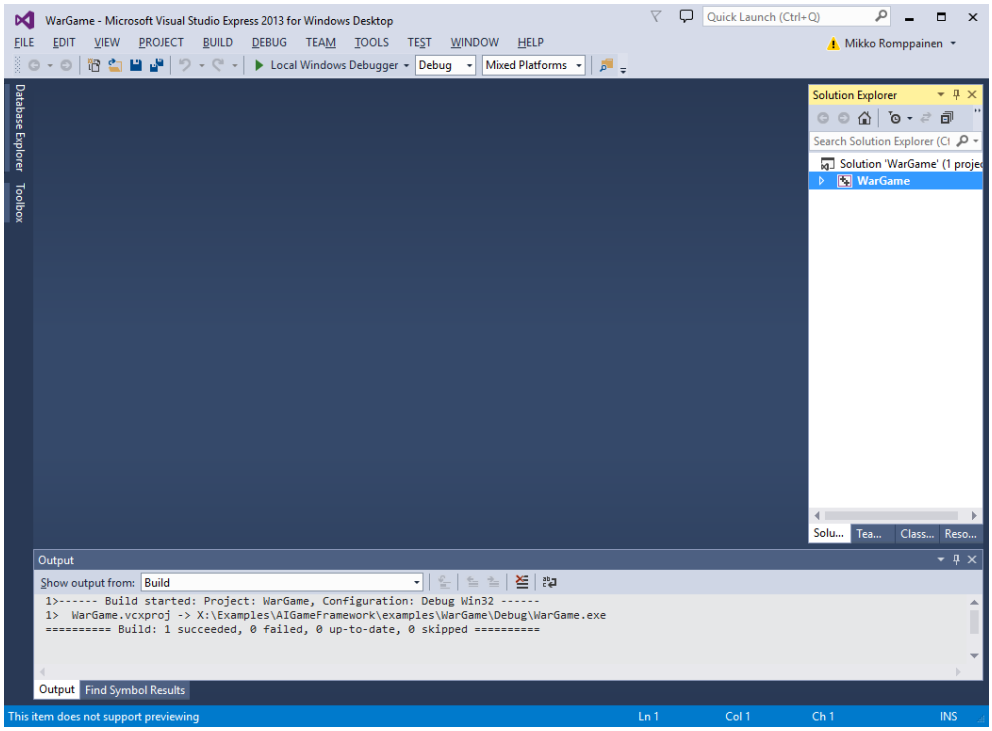
You need to make section **"Making of static library"** only, if you have not yet created it. When you have fully operational AI ready, then you need to read section **"Making of release packet**". In order to pass the course, you need to beat "PassCourse" AI from examples folder. Check that class for reference. Also in this document is **"FAQ Section"** theck that also. And **Before you begin, read this manual each page! :)**

Making of static library project

Open latest version of WarGame from WarGame.sln file .



Rebuild wargame:



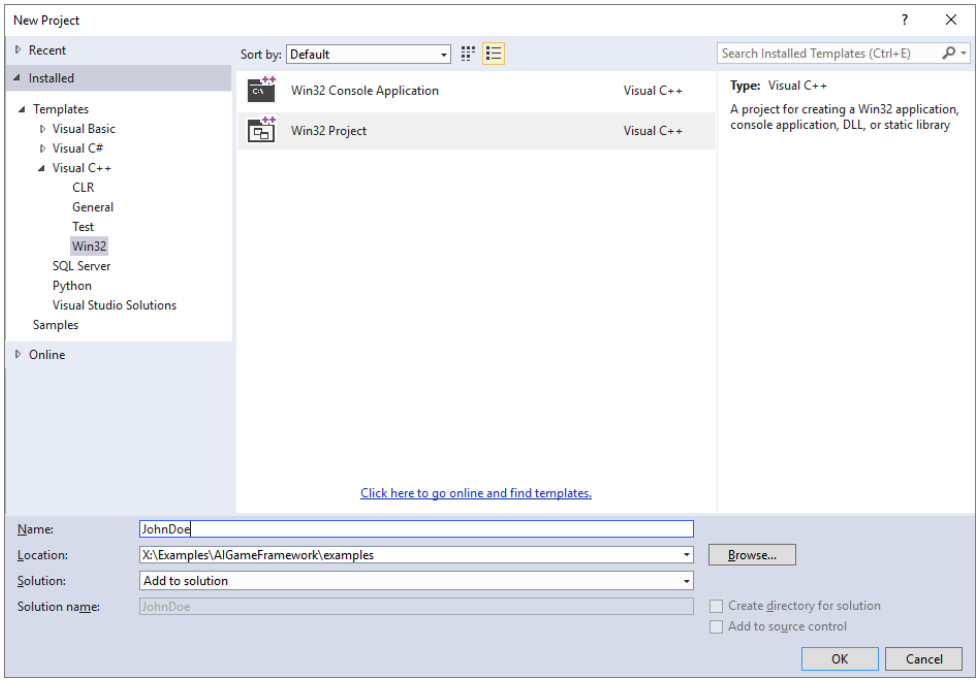
Add new static library project

Select from Visual Studio Menu: **File -> New Project...**

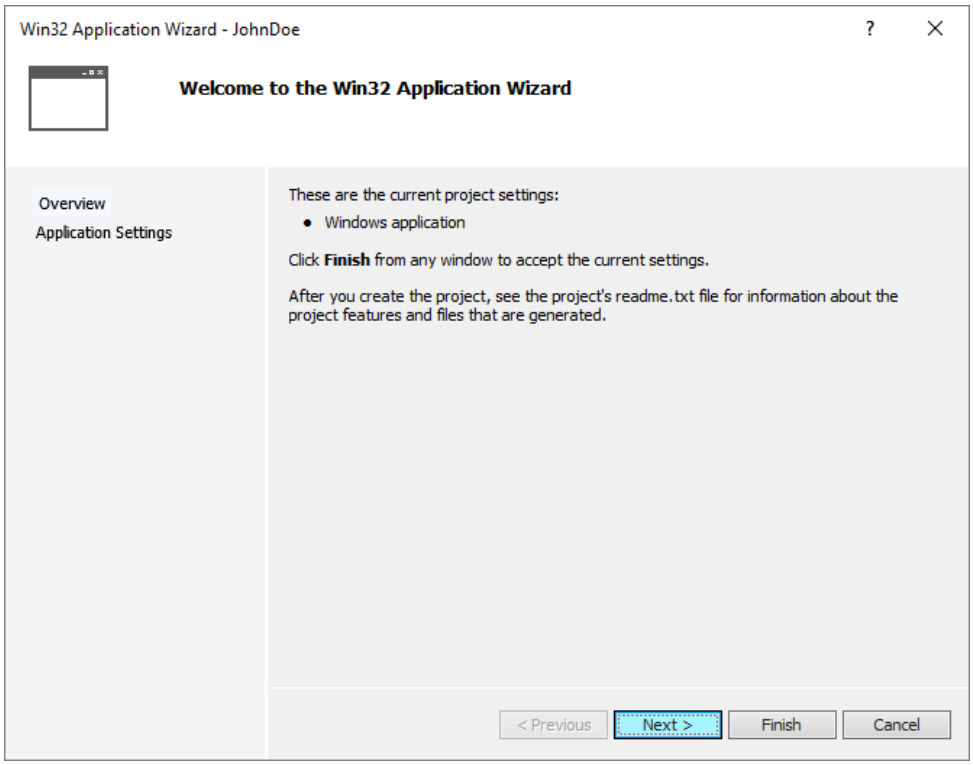
Name: **JohnDoe**

Location: **"AIGameFramework\examples"**. Please browse correct path from your computer.

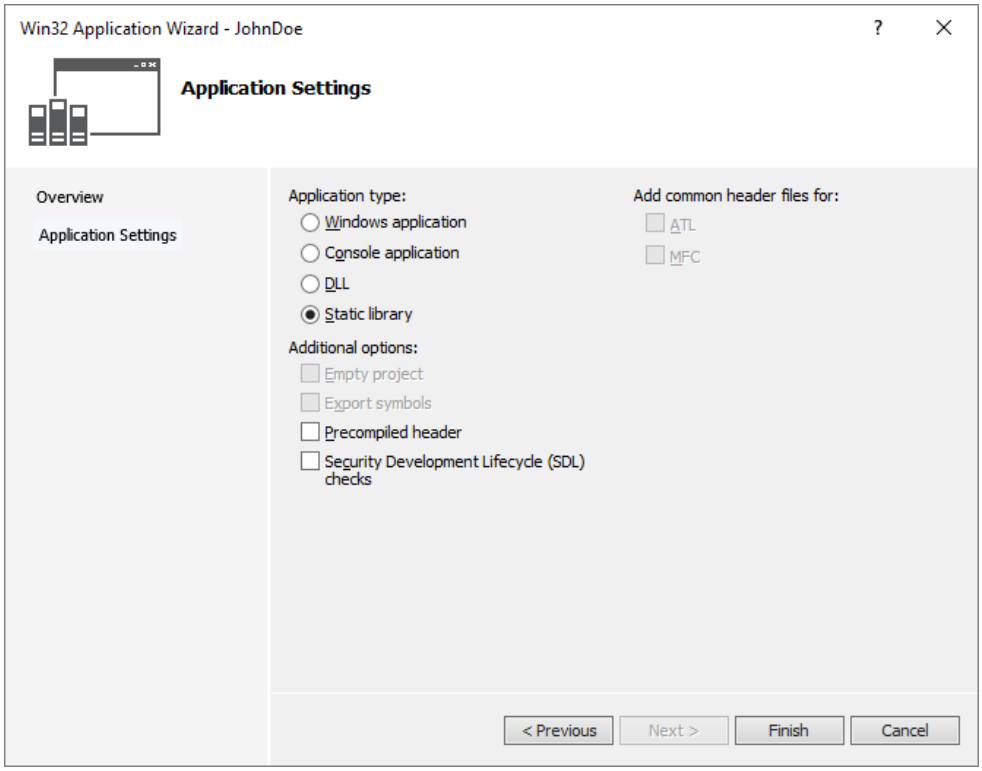
Solution: **Add to solution.**



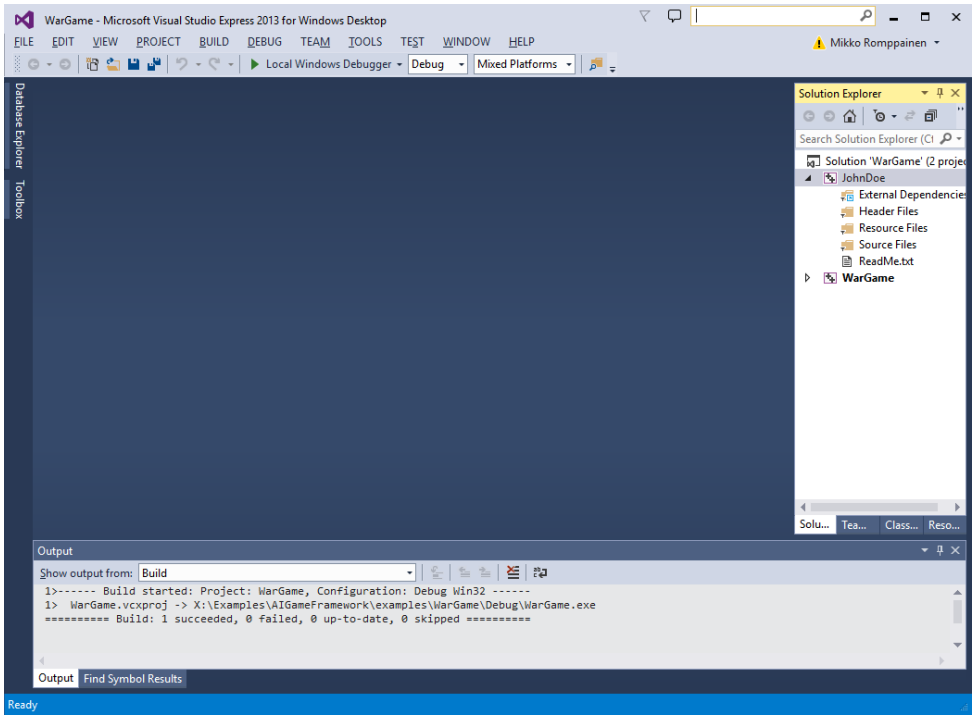
Press **Next**.



Select **Static Library** and **not using Precompiled Header** also **check out SDL checks**

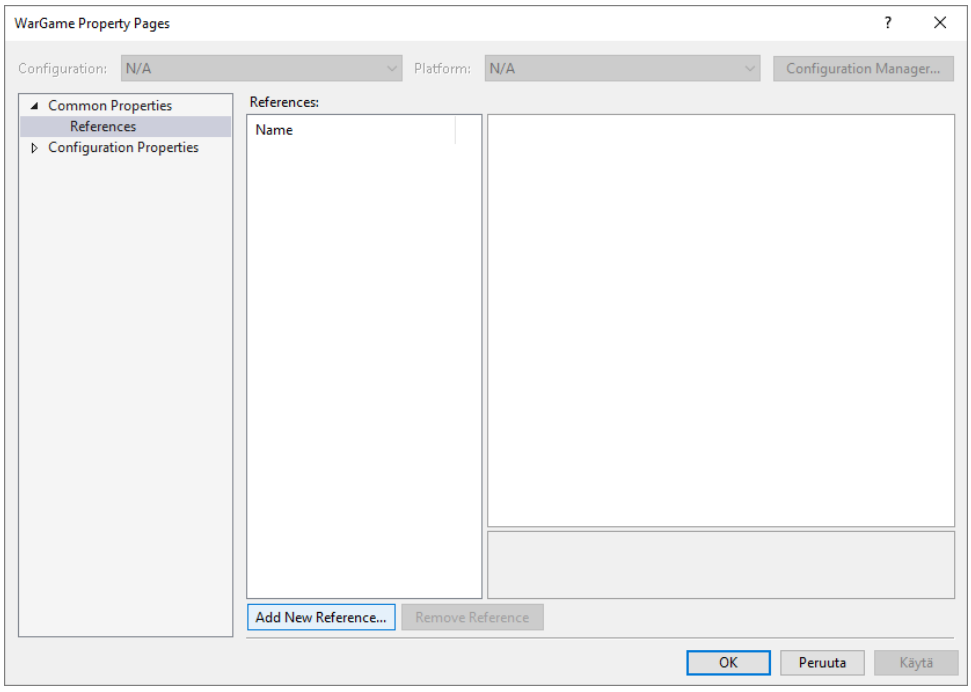


Press **Finish**.

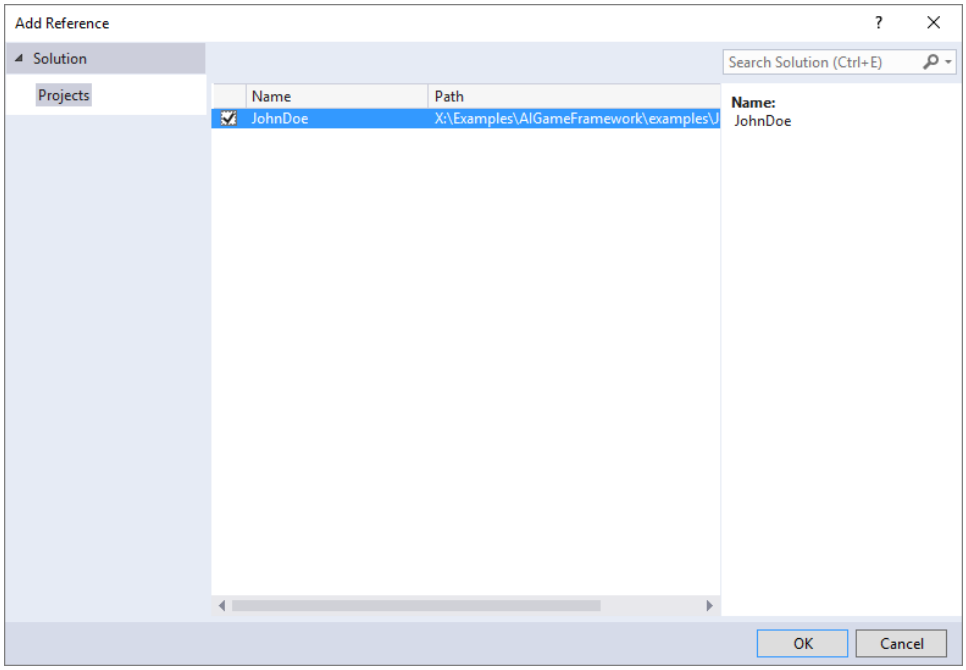


Add project references

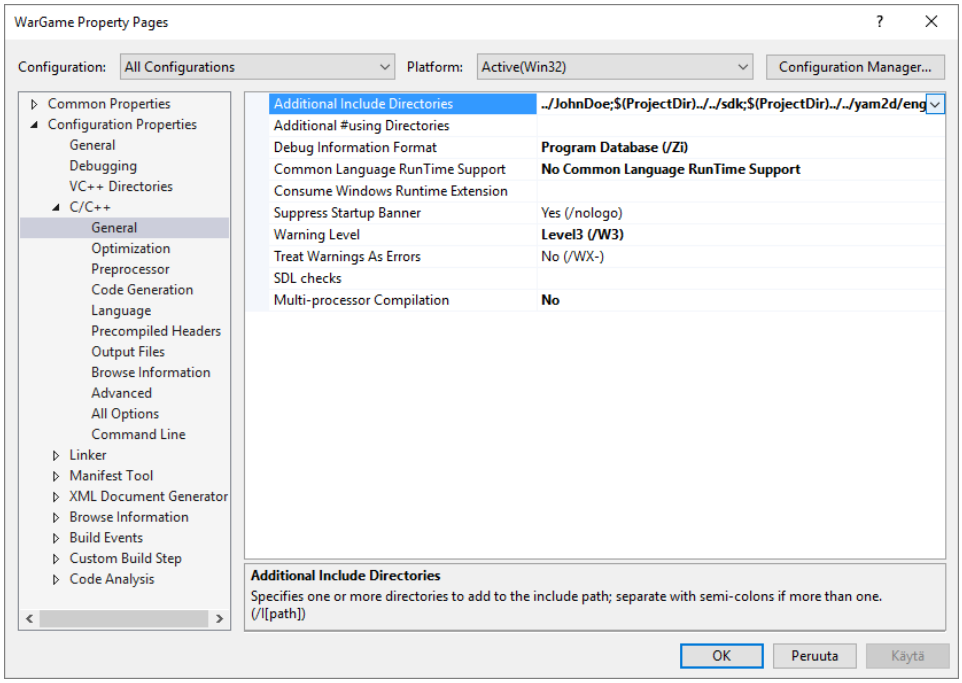
Now, we need to add correct reference to WarGame-project to be pointing to JohnDoe-project. **Right click WarGame** and select **Properties**. Select **Common Properties -> References** from menu on left.



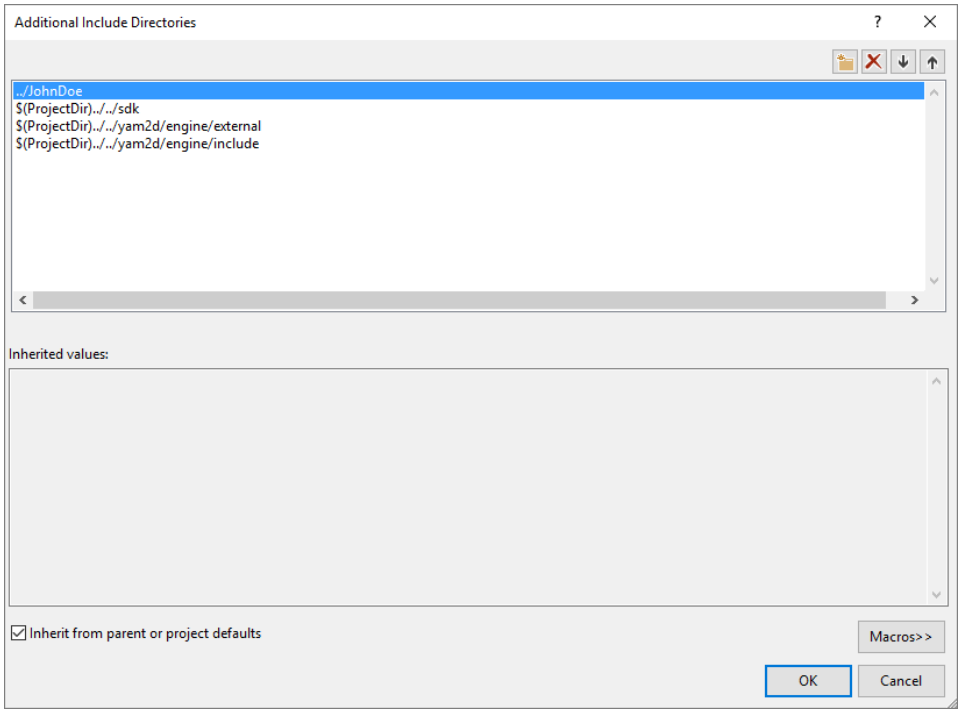
Click **Add New Reference**. Add new reference to JohnDoe-project. **Click OK**.



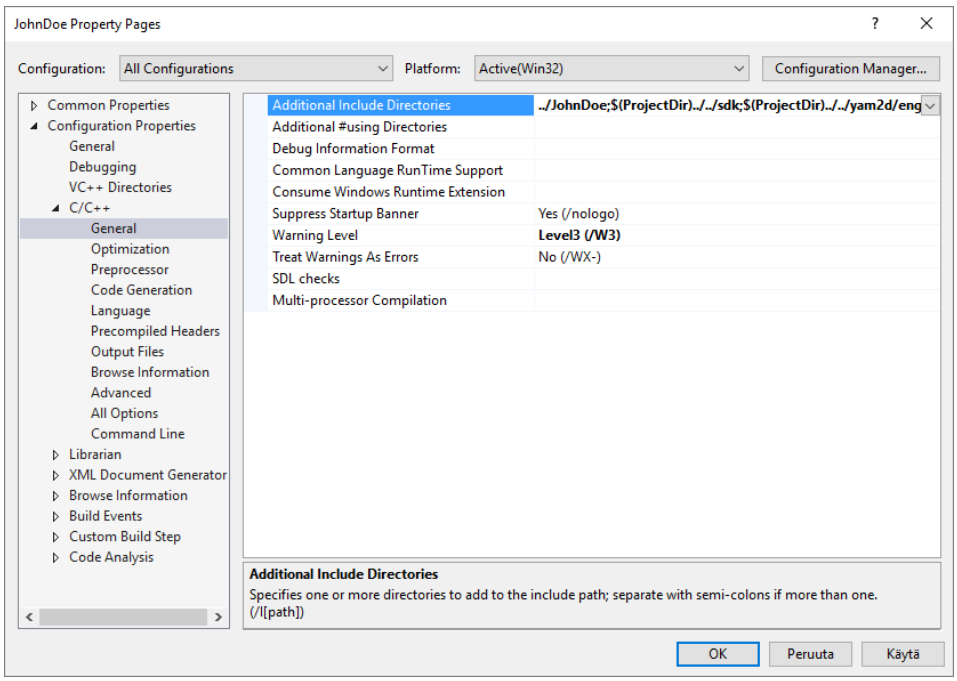
Add **Additional Include Directory** from **Configuration Properties -> C/C++**.



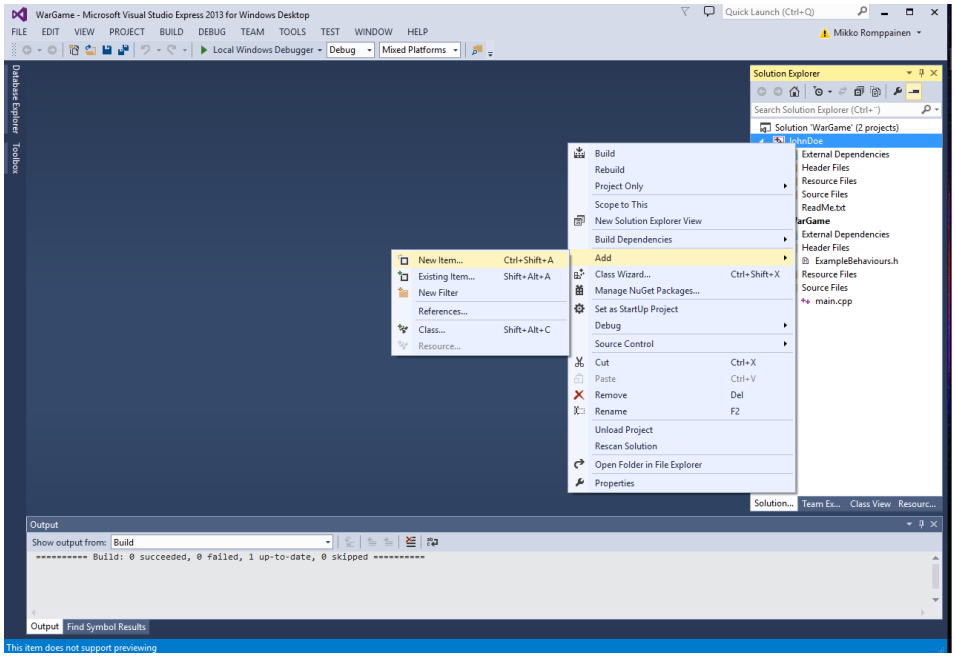
Add "..\JohnDoe" to additional include paths and click OK:



Copy additional include paths from WarGame-project and **paste it to JohnDoeProject additional include paths**:

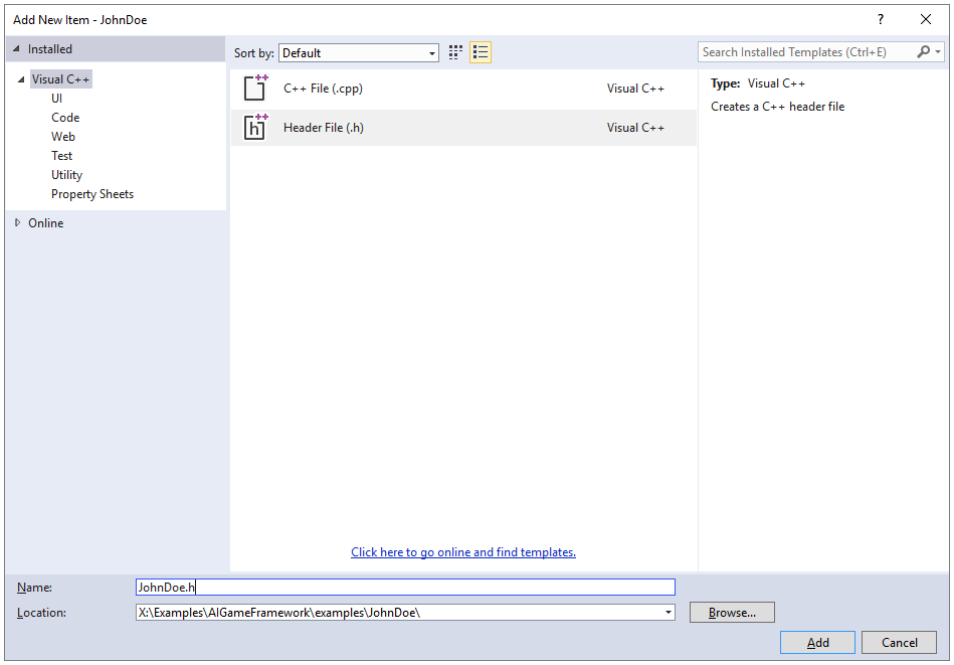


Click OK on **Project Properties** dialog. Now **add two new files** to the JohnDoe-project:

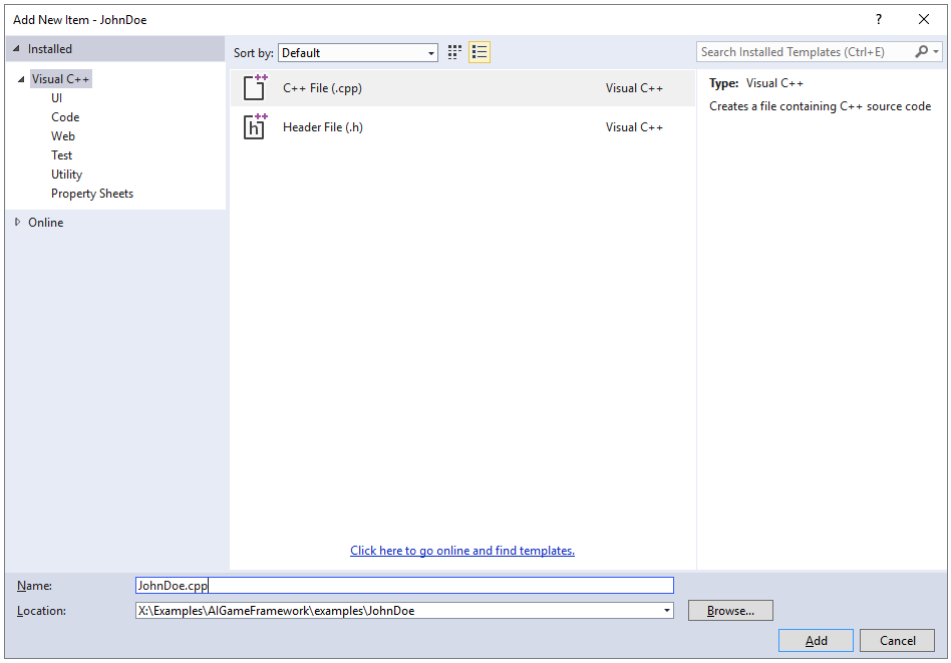


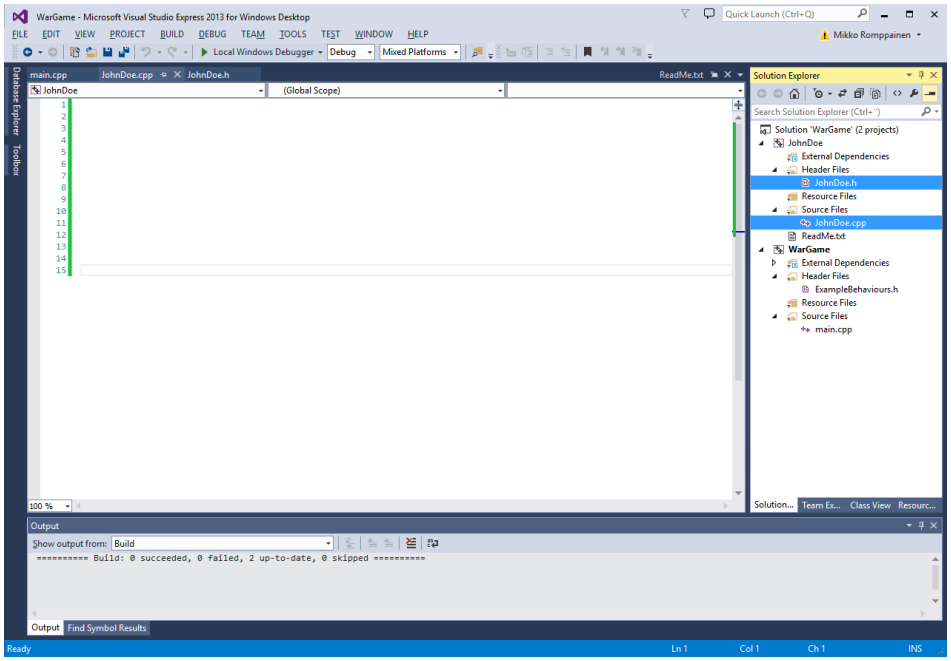
Add new source files

Add **JohnDoe.h**



Add **JohnDoe.cpp**





Rebuild WarGame. It sould compile okay. Now you need to modify some files:

JohnDoe.h:

Copy-paste classes: **MyAI** and **MyPlayerController from WarGame main.cpp** to **JohnDoe.cpp** of **JohnDoe-project**. You need to silghty modify include-files**.** Also you need to **rename classes**: **MyAI -> JohnDoeController** and **MyPlayerController -> JohnDoePlayer**. Also **change component name** to be correct in **createPlayerCharacterController**. Also add both classes under namespace JohnDoe.

See JohnDoe.cpp from examples for absolute minimum file inorder to compile.

Note! You need to put also ALL OF YOUR EXTRA CODE (like path finding stuff) under JohnDoe namespace for avoiding conflict with other students class names during link (this will be fixed in the future).

For **JohnDoe.h** file, you need to **add a createNewPlayer-function** and **JohnDoePlayer-class forward declaration** under **namespace JohnDoe**. **See JohnDoe.h from examples**. You need to also **implement createNewPlayer-function to JohnDoe.cpp**. **See JohnDoe.cpp** for a reference.

For main.cpp in WarGameProject, you need to **delete old classes** and **includes**. Only **include GameApp.h and JohnDoe.h**. Also, you need to **modify setDefaultGame call to match names to your component (JohnDoe)**. Also you need to add **call to JohnDoe::createNewPlayer()**, in **app->setPlayer1Controller** and a**pp->setPlayer2Controller** calls. See **main.cpp** from **WarGameStatic-project** from examples.

Try **rebuild solution**. After everything rebuilds okay, you can continue to "Making of release packet".

Note! If you have added some extra classes to example project, please add those classes to JohnDoe-project. Typical code might be something like path finding stuff etc your own code. Remember to check, that all code which is compiled in JohnDOe-project, is under JohnDoe namespace (or some other non commonly used namespace). This is for avoiding the namespace conflict between each student code, which might be caused during linking of "tournament executable".

Making of release packet

Make sure that you have latest released version of AIGameFramework. You must be able to recompile the **JohnDoe**-project in both, **Debug** and **Release** configurations. **Rebuild JohnDoe for Debug and Release. Copy** files named **JohnDoe.lib** from folders $(SolutionDir)\$(Configuration) to following kind of folder structure, where JohnDoe-folder is root folder. **Copy also JohnDoe.h -header file to JohnDoe-folder**:

JohnDoe

|- Debug

|- JohnDoe.lib

|- Release

|- JohnDoe.lib

|- JohnDoe.h

After copying libraries to correct places, select JohnDoe-folder with mouse right click and Add it to JohnDoe.zip. Email zip.file to: mikko.romppainen@kamk.fi with subject: "JohnDoe version x.x", where x.x version is version number (you can decide the version number).

Friendly Asked Questions

How can I access level map data?

You can ask it from GameEnvironmentInfoProvider for example in onGameStarted:

```
// Get speed map...
AIMapLayer* moveSpeedMap = environmentInfo->getAILayer("GroundMoveSpeed");
```

How can I get "walkable cost" of the (GameObject) position?

Depends, if you need to go trought all pixels (tiles):

```
for (size_t y = 0; y < moveSpeedMap->getHeight(); ++y)
{
for (size_t x = 0; x < moveSpeedMap->getWidth(); ++x)
{
// Get pixel from pos x,y. Pixel values are 4 bytes reach color channel (RGBA).
// Pixel in current implementation is
white=[255,255,255,255]/gray=[127,127,127,127]/black=[0,0,0,0])
// Value = 0 -> Pixel not walkable (0.0x speed)
// Value = 127 -> Pixel slowly walkable (0.5x speed)
// Value = 255 -> Pixel fully walkable (1.0x speed)
uint8_t red = moveSpeedMap->getPixel(x, y)[0];
uint8_t green = moveSpeedMap->getPixel(x, y)[1];
uint8_t blue = moveSpeedMap->getPixel(x, y)[2];
uint8_t alpha = moveSpeedMap->getPixel(x, y)[3];
}
}
```

If you need to access pixel by game object position:

```
AIMapLayer* moveSpeedMap = environmentInfo->getAILayer("GroundMoveSpeed");
uint8_t value = moveSpeedMap->getPixelFromPos(gameObject->getPosition())[0];
```

How can I make my own debug layers?

You just need to call getAILayer of GameEnvironmentInfoProvider. Just add some custom name for the layer. You might want to also call setLayerOpacity in main.cpp for GameApp to control layer transparency. After creating AIMapLayer, you can use setPixel to set pixels to the map.