# Assignment 1 – The Village of Testing

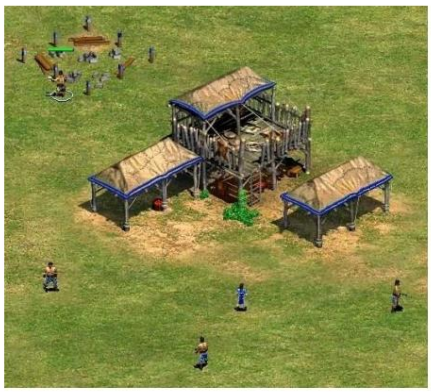

Image source: https://www.softpedia.com/reviews/games/pc/Age-of-Empires-2-HD-Edition-Review_344425.shtml

Your task is to create a series of functions and classes that simulate a growing village. Each worker is registered as an item, and they have a limited amount of resources to make buildings with. You must create functions and then test them to see that they do what they are supposed to do.
The application must be developed with C# or Java, and then tested with any unit testing tool.

The goal is for the community to grow large enough to build a castle. For this they must collect resources, build buildings and not die of starvation at the same time. The program must be playable: Make sure that a user in the main function is put in a loop and can call most of the functions in the Village.
AddWorker(), AddProject() and Day() in particular. You do not need to test the main function.

There should be a Village class, a Building class and a Worker class.

The main class shall be Village. It is the functions of this class that we want to test. It must contain

- A variable for how much food the village has
- A variable for how much wood the village has
- A variable for how much metal the village has
- A list of all the buildings in the village
- A list of all construction projects that are not yet finished
- A list of all the workers in the village
- A variable for how many days have passed since the village was founded

Leave the variables private, but for the sake of testing, make sure there is a function to retrieve each of these values. GetWood(), GetWorkers(), etc, and these are public.

Make the Village constructor create three houses and give the village 10 food. They will need it.
The rest of the values can be empty lists and the value 0.

Create a class Worker. Each Worker must have a name, a string that describes what they work with and a delegate/functional interface that contains what happens when they work.

The Worker class has a function "DoWork". This calls a delegate/functional interface that is created as an object in the Worker. This should call one of four functions in Village.

Village has one function per possible thing a Worker can do. “AddWood”, adds 1 wood to village resources. "AddMetal", which does the same for metal. "AddFood", which provides type 5 food to the village. More than 1, anyway. Then "Build", we get to what it does. Send in via a lambda expression which should be done when a worker is created.

Village must have a function called "Day()". The Day function must call DoWork() on all Workers. Besides that, it must feed all the workers. Worker should have FeedWorker() function.
For each Worker, 1 food is subtracted from the village's resources. If it happens that the village does not have enough food, set a boolean "hungry" to true in the Worker on those who did not get food. Add a check to DoWork() that makes a worker only work if they are not hungry.

Add a counter "daysHungry" and a boolean "alive" to the Worker. If they get hungry, daysHungry is incremented by 1. If they ever get food, set daysHungry to 0 and hungry to false. But, if daysHungry reaches 40 or higher, set alive to false. Make sure that if alive is false, you can neither feed the worker nor make it work.

Have a separate function BuryDead() that removes all Workers that have "alive = false" from the Workers list. If the list of Workers now becomes empty, print "Game Over".

We will want to build buildings for our village. Create a class Building. It must have a string name, then a boolean is if the building is finished or not. It must have two numbers; one is how many working days it takes to complete the building, and the other how many working days have been spent on the building. This, plus each building must have how much it costs in wood and metal to start construction.

When a worker runs the Build function, it should enter the list of projects, take the first one, and add 1 to the number of workdays spent on the build. If the number of working days spent exceeds the number of working days it takes to build the building, then the building is complete. Move it from the list of ongoing projects to the list of finished buildings. Depending on the building, the clear building should have an impact on the village, which should be doable with variables.

Buildings:

House – Costs 5 wood to start, takes 3 days to build – Two workers are allowed to exist per House that exists. Make sure the Village has an AddWorker() function, and it first checks if there are enough houses before adding a new worker.

Woodmill – Costs 5 wood and 1 metal to start, takes 5 days to build – Makes a worker gathering wood pick up 2 more wood per day. You may want to change a variable used in AddWood() when Woodmill is finished.

Quarry – Costs 3 wood and 5 metal to start, takes 7 days to build – Makes a metal gathering worker pick up 2 more metal per day. You may want to change a variable used in AddMetal() when Quarry is finished.

Farm - Costs 5 wood and 2 metal to start, takes 5 days to build - Makes a food gathering worker pick up 10 more food per day. You may want to change a variable used in AddFood() when the Farm is done.

Castle – Costs 50 wood, 50 metal and 50 days to build – When the castle is complete, a small message is written “The castle is complete! You won! You took ” + days + “ days to complete the castle.

## Testing

I want you to test every feature in the Village.

AddWorker() – A function that allows one to add a new worker, but only if there are enough houses for them. Takes a delegate/functional interface for which function you run afterwards.

1. Test if you add a worker. Then two. Then three. Assert that there are as many as it should.
2. Test if trying to add a worker but there are not enough houses.
3. Test if the right thing happens if you create a new worker with a function and then call Day().

Day() – The Day function should loop through all your workers, feed them and then call DoWork() on each one.

1. Try not having any workers and running Day().
2. Try adding some workers and running Day() while you have enough food.
3. Test Day() but there is not enough food, make sure the right things happen.

AddProject() – Adds a new building to the list of buildings to build. It should only do this if you have enough resources, and if you have, these should be deducted from your total resources.

1. Try adding a building that you can afford. Test that it works and that the right proportion resources are drawn.
2. Try trying to add a building you can't afford and make sure it doesn't works.

Be sure to test the different specifics that apply when different buildings are completed!

1. First run Day() one day before a WoodMill is ready. See that you get 1 wood first, then more wood Day() after the building is finished. The test needs a worker who makes firewood and one who builds.
2. Do the same kind of tests for food and metal.
3. Try to start the building of a house, then put some workers on it, and then that right number of Day() functions will be finished with the project.

Check that the hunger mechanic works as it should!

1. A worker who has not received food does not work. Make sure it works.
2. A worker who has not received food for 40 days gets "alive = false". Make sure it works.
3. It should not be possible to feed a worker with "alive = false".
4. Check that the "BuryDead()" function removes those that have "alive = false".

Finally. Start from the beginning. Run all functions, a combination of AddWorker(), AddProject() and Day() until a slot becomes ready. List how many days it took to finish. Test if it took the number of days it should have.

## For VG

Create two new features in the Village. Call it SaveProgress() and LoadProgress().

Create a new class. Call it DatabaseConnection. It has a Save function and a Load function. Save takes ALL variables worth saving, and should save them to the database... although you don't have to actually implement it. Load() is a function that does not return anything, but it loads the values from a database into its own variables. You should then be able to retrieve these individually with GetWood(), GetFood(), GetWorkers(), GetBuildings(), GetDays(), etc... although none of these functions must be implemented.

Create an object of type DatabaseConnection in Village and implement SaveProgress and LoadProgress. SaveProgress passes all relevant variables to a function in DatabaseConnection and LoadProgress first runs DatabaseConnections Load() and then individually populates all variables with GetWood() etc.

Test LoadProgress(). Use mocking to give directly what should be returned from various functions in DatabaseConnection. Try using LoadProgress to load the values, given all the values it is that will be loaded, and then do whatever it takes to prove that you actually got the values we wanted.

Besides this, create a function AddRandomWorker(). This uses a random generator to create a new worker who gets to decide for himself which work he will work on. So a different delegate/functional interface depending on the result of a random generator.

Test AddRandomWorker(), but use mocking so that, in our test, the result from the random generator will always be the same. Then run Day() for our RandomWorker to do its job and then test that it turned out right.

Good luck!