Concurent assignment #2

Submit Assignment

Due Dec 18 by 11:59pm **Points** 15 **Submitting** a file upload **File Types** zip
Available Nov 29 at 12am - Dec 18 at 11:59pm 20 days

Less Fun Version of Warcraft 3

Introduction

Warcraft 3 us a strategy game in which opposing factions fight to determine who will be the winner, after completing the necessary preparations. The assignment is to simulate a similar game, but much simpler. In the Base Task there is no need to create two factions and start a war, only to create our own base. The Second Task requires us to implement the war phase of the game.

The **Building, Resources, Simulation, Unit, UnitType** classes do not need to be modified while completing the tasks.

Base Task – 5 points

The **Simulation** class is the entry point of the simulation, where we create a **Base** object. **Base's** constructor requires one parameter; the name of the base.

The goal of the base is to create certain buildings, personnel. Both personnel and buildings have some common attributes; **goldCost**, which determines how much gold is needed to create that unit, **woodCost**, to indicate the required amount of wood for the unit creation, **foodCost**, which determines how much food is needed to keep the unit alive (buildings don't need food, this attribute is 0 for them), **owner**, which is a reference to their **Base**, **unitType** to determine the type of the unit, and **buildTime** which describes how much milliseconds is needed to create the unit. The type of a unit is determined using an Enum, **UnitType,** which has the following information about a unit; **goldCost, woodCost, foodCost, buildTime.**

UnitType has the following values:

Name	**Gold Cost**	**Wood Cost**	**Food Cost**	**Build Time (ms)**
FARM	80	20	0	2000
LUMBERMILL	120	0	0	4000
BLACKSMITH	140	60	0	5000

PEASANT	75	0	1	1000

The **Resources** class helps the **Base** to keep track of the resources (**gold, wood**) and how much people is it fit to hold (**capacity, capacityLimit**). The **capacity** determines how much food is needed to upkeep the current units, **capacityLimit** shows the maximum amount of personnel we can upkeep. **Capacity can never surpass the capacityLimit!** The starting value of **capacity** is 5.

The **Peasant** class represents the workers. It is a subclass of **Unit** has the following methods, which need to be implemented;

- **startMining:** Starts mining. Every100ms 10 gold is harvested.
- **startCuttingWood:** Starts cutting wood. Every 100ms 10 wood is harvested. **A peasant cannot cut wood and mine gold at the same time.**
- **tryBuilding:** Tries to build a building if there are enough resources to do so, and if the peasant is free currently. A worker cannot start building while its harvesting, and cannot start harvesting while its building something.

The **Building** class represents the buildings. *Lumbermills* and *Blacksmiths* don't affect the gameplay at all. Every *Farm* grant 10 more food, thus increasing the capacity limit in the **Resources** class.

In the constructor of the **Base** class create 5 **Peasants.** 3 of these 5 workers should mine gold, one should cut trees, and one shouldn't do anything. The Base class has a **startPreparation** method which aims to achieve the following;

- Create 10 **Peasants**. 5 of these 10 should mine gold, 2 should cut wood. There cannot be more **Peasants** than its allowed by the capacity limit. At once only one **Peasant** can be trained.
- Create 3 *Farms*.
- Create 1 *Blacksmith.*
- Create 1 *Lumbermill.*

After completion, the workers should stop harvesting. As soon as there are sufficient resources to build something or create a unit, that should be started right away.

Second Task – 10 points

The new **Personnel** class is a subclass of **Unit**. This **Personnel** is the new superclass of **Peasant** and the newly created **Footman** class. Every **Personnel** has **health,** minimum and maximum attack points (**attackMin, attackMax**), and an **opponent** who is another **Personnel**. Implement the following methods:

- **startWar:** Receives the enemy **Personnel** from it, and starts attacking it until either of them drops dead. If the opponent has lost his health, the **Personnel** proceeds so select a new **opponent** until there enemy army is obliterated. A **Personnel** can attack between 100-200ms, and deals a random damage within a range to its **opponent**. **Note:** It is not necessary for a **Personnel** to attack back those who attack him. While A attacks B, B can attack C.

- **loseHealth:** Loses health after being attacked. If **health** is 0 or less, the **Personnel** is to be removed from the army, and capacity should be adjusted (decreased) in **Resources**.

Personnel type	Health	Attack Min	Attack max
PEASANT	220	5	6
FOOTMAN	420	12	15

Base's startPreparation changes so that now a *Barracks* needs to be built as well, and 10 footmen trained. In order to be able to train **Footman**, the **Base** must have at least 1 *Barracks*.

Name	Gold Cost	Wood Cost	Food Cost	Build Time (ms)
BARRACKS	160	60	0	6000
FOOTMAN	135	0	2	1500

After **startPreparation** has finished, **assembleArmy** is invoked which creates the army of a base – This needs to be implemented as well. The army should consist of every **Footman** and **Peasant** (after they are done with their work).

Another, enemy base is created in the simulation. Both bases first prepare, then assemble an army, and then launch an attack to each other. This is done via invoking the **goToWar** method, which needs to be implemented. This method can only be invoked once **assembleArmy** has happened already on a base. The **goToWar** method is responsible to launch a bases attack against the other base. Every soldier should be represented as a separate thread, so they aren't waiting for each other to attack. When the war is over – meaning one of the armies is empty – the winner of the war is announced. Both bases print if they won or lost the war.

secondassignment_student.zip