ELEC-A7150, C++ Programming

Documentation

The Settlers

Group 2

Lehto Joonas
Löfstedt Arthur
Malinen Joonas
Nieminen Tarmo

1. Overview

The goal for this project was to create a simplified version of the classic real-time strategy game The Settlers. In the original game, the player begins with some basic buildings and a modest amount of settlers, with which to build up a flourishing colony. The player can give various commands to these settlers, who then fulfill those commands independently. Examples of commands include "collect wood", "build a building" and "defend fortress". The real-time aspect of the game means that carrying out these given tasks takes a certain amount of time, as does moving around the map. The original game was quite complex and had a large amount of different buildings and commands.

In our version of the game, the player begins with a Warehouse, used for storing gathered resources and manufactured items, and a House, containing two Settlers. The player can then click on a settler and command him/her/it to, for example, chop wood. The game starts with the warehouse containing no resources, so this might be a prudent choice of task. More settlers can be created by building a new house. The settlers are created before the house is finished, so the created settlers themselves can be used to build their own house.

The map is statically created, so the game always starts with the same map, buildings and resources. The way the map and default buildings are read into the program, however, means that a way to choose between different maps could easily be implemented in the future. The map is a grid of squares, where each square represents a different type of terrain. There are five different types of terrain: basic grass, stone, mountains, beach and water. The buildings and trees are then added on top of this terrain map, with settlers visible above all, to help with clicking on a settler. For technical reasons, trees are considered to be buildings in this game. The players, and settlers, actions affect the map: buildings and roads can be built and trees can be depleted and disappear. Quarries, or "stonecutters", and Mines are used for collecting stone and iron respectively, and these buildings also disappear after depletion.

There are eight different types of buildings in the game, although the functionality for some of them has yet to be implemented.

First of all, the humble House. The house is used as a way to create new settlers, since every time a house is created, two new settlers are created with it. In addition, after completing a building, settlers seek out the closest house to hang out in and chill while waiting for a new constructionsite to appear. For a while during development, settlers would seek out the closest tree instead, which was pretty amusing!

The second type of building is the Warehouse. This building functions as an inventory for the resources settlers gather, and resources used in building new buildings are taken from the warehouse.

The third building is the Stonecutter, which functions as a resource gathering spot for stone, and needs to be built on top of rocky terrain. Similarly to trees, this building gets destroyed when it is depleted, and needs to be rebuilt.

Fourth, you have the Mine. This structure is used for gathering iron from mountains, and consequently needs to be built on top of a mountain. Like the stonecutter, this building also disappears after a while. The mine, however, is special compared to the other buildings covered so far: it is the first building in our list which not only requires wood to be built, but also stone!

Fifth, we have the Blacksmith. This building requires wood, stone and iron to be built. The blacksmith is used for converting iron into swords. The blacksmith is somewhat buggy at the moment.

The sixth building is the Keep, which is extremely expensive to build. Unfortunately the building currently only acts as a status symbol.

Finally, seventh on the list is the Road. Roads are special, since they don't require any resources to be built, nor do they require a settler to build them. They are simply placed into the terrain by the player. Roads function as the preferred path of moving for settlers.

2. Software structure

2.1 Logic structure

The logics structure contains several different classes.

Coordinates is used to locate other objects on the map and to compare different objects. The methods it has are all focused on returning and changing coordinates.

Terrain is the base piece of the map, where everything is built and settlers use for movement. Terrain has functions for adding and removing buildings. These functions change terrains private parameters to match the building that occupies the terrain.

Map creates and sets the map for the game. The functions in this class focus on finding wanted terrains. Map also contains the path finding algorithm solvePath, which finds the shortest location between to given coordinates. The wanted map is located in map.txt, where it is read by setMap.

Settler class is the class for the games units, settlers. These settlers have the information about their hp and inventory. Inventory contains from two dimensioned vector where first one is a vector that contains the settlers inventory and second one is an integer that tells you how many items the settler can hold. The settlers only know the id of their task, not how to do it. Settlers also has function to access their inventory. These functions are called by game later. The settlers move function makes the settler move according to the path solvepath made for them.

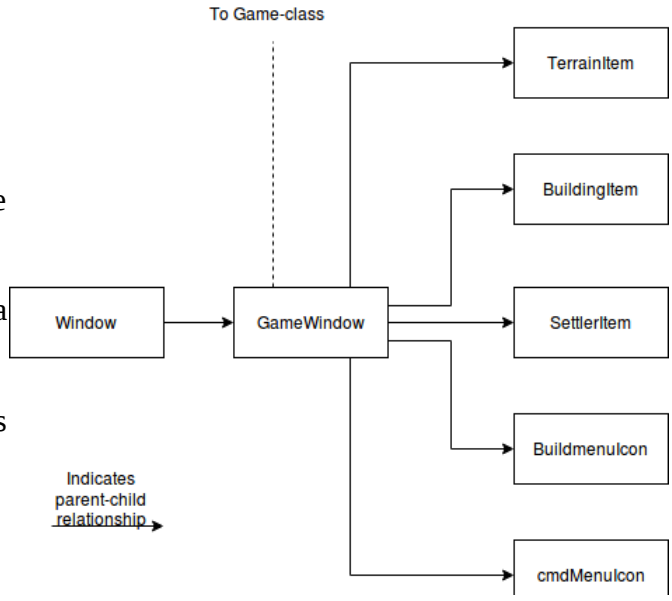
Buildings is an abstract class for other buildings to use. Building class has functions for building the building, managing its inventory and managing hp.

House, warehouse, tree, mine, stonecutter, blacksmith, keep and roads are sub-classes for building. These classes have all different constructors that tells the building what items are required for their building. Addition to this House has settlers living in them. Tree is the only building that you can’t build. Trees take damage when harvested and when destroyed, removed from the game. Stonecutter and mine act the same, with the difference that they must be built.

Game class is where the magic happens. This class contains all the methods necessary to run the game. setBuildings reads the buildings from default.txt for the beginning and sets them on the map. Game contains methods for adding buildings and settlers. These are both implemented by calling different methods from settler and different buildings classes. The biggest part of the game class is simulate and different tasks. Simulate is used in the beginning of every “turn” and it determines what the settler does. This function and function used to help it replaced the TaskHandler originally planned. Simulate goes through all the settlers in reverse order with the enemy being called upon last. It determines whether settler has delay left or can it act. If it can act it calls settles task and calls the corresponding task. These tasks contain checks for settlers location and inventory and if the requirements are correct, it completes the task.

2.2 GUI structure

The Graphical User Interface, or GUI, and the graphics for the game, are created with the help of the Qt graphical framework. The main structure of the GUI consists of two windows. The first is the "Main Menu"-window, which is a "Window"-class object, and is created in "main.cpp". The implementation for this class is in the "mainwindow.cpp"- and "mainwindow.h"-files. The "Window"-class is a "QWidget", and inherits that class. The "Main Menu"-window includes three "QPushButton"-class objects, which create the three buttons used in this window, and a "QGridLayout"-object to place the buttons in. Objects in Qt have a parent-child relationship, where the parent is responsible for the child. This means that closing the parent also closes the child, but closing the child does not close the parent. Qt handles memory management in an own way, which does not seem to work well together with Valgrind, since even just opening a simple window creates a memory leak of roughly 2000 bytes, according to Valgrind. In the case of this "Window"-class, the pushbuttons are children of the layout, and the layout is a child of the window. These buttons are connected to the two slots, or methods/functions, of the "Window"-class, meaning that clicking a button runs the code in the slot it is connected to. A button can also be connected to an external or inherited slot, for example the "QUIT"-button which is connected to the close()-method of the "QWidget"-class. The "NEW GAME"-button is connected to the NewGame()-slot, which, when executed, creates a new object of the "GameWindow"-class, with this "Window"-class object as parent.



The "GameWindow"-class also inherits the "QWidget"-class, but it is also specifically flagged as a "Qt::Window", so it opens up a new window instead of existing inside the borders of the "Main Menu"-window. The "GameWindow"-class is the structural heart of the GUI, since it

connects to most other things, most importantly the "Game"-class. The actual game is created when the a new "GameWindow"-class object is created by the "Main Menu"-window. Through the "Game"-class, "GameWindow" gains access to the rest of the classes needed for the game to be presented graphically. Like the "Window"-class, the "GameWindow"-class is the parent of several other Qt-objects. "GameWindow" contains three "QGraphicsScene"-class objects, three "QGraphicsView"-objects, a "QGridLayout", a "QPushButton", two "QLabel"-objects, and a "Qtimer"-object.

The "GameWindow" is the parent of "scene", "buildscene" and "commandscene". These "QGraphicsScene"-class objects are themselves parents of different "QGraphicsItem"-class objects. "scene" contains the graphical representation of the map, buildings and settlers. In accordance to this, "scene" is the parent of a bunch of "TerrainItem"-, "BuildingItem"- and "SettlerItem"-class objects, all of which inherit the "QGraphicsPixmapItem"-class. Except for "BuildingItem", the other two also inherit "QObject", to be able to utilize the "signals and slots"-aspect of Qt. Being able to emit signals makes "TerrainItem"- and "SettlerItem"-objects clickable. Similarly, "buildscene" is the parent of "BuildmenuIcon"-class objects, which work in the exact same way as "TerrainItem"- and "SettlerItem"-objects. Lastly, "commandscene" is the parent of "cmdMenuIcon"-class objects, which also work the same way as the ones above.

The .png-imagefiles used for the graphics are stored in the same directory as everything else. For the program to be able to access these files without knowing the precise location of the files, a resource management file, "media.qrc", was created. This .qrc-file is an XML-based directory, storing all the filenames for the imagefiles under a "/graphics"-prefix, telling the "Makefile" that these files exist in the same directory as the "Makefile" itself. To access these files in the code, one simply needs to type ":/graphics/filename.png", instead of typing out the whole path.

3. Instructions for building and using the software

3.1 Building

This guide assumes the player is using a Linux based system. Download the software any way you prefer. When that is done, navigate to the “src” folder. Now, open this directory in the terminal and simply type “make”.

If this for some reason results in any errors, delete the “Makefile”-file and type “qmake main.pro” in the terminal. This recreates the Makefile. You should now be able to run “make” without errors.

When that is done, you can run the software either by typing “./launch” in the terminal or double-clicking the “launch”-file.

3.2 Using the software

The first thing you see when launching the program is the "Main Menu". This menu has two buttons, "QUIT" and "NEW GAME". From here you can, correspondingly, either close the program by pressing the "QUIT"-button, or start a new game with "NEW GAME".

Clicking on the "NEW GAME"-button opens a completely new window, called the "Gamewindow". This new window is opened on top of the previous window, so, if you want, you can access the Main Menu by simply minimizing the Gamewindow or dragging it out of the way. The recommended way to access the Main Menu, however, is to click the "Main Menu"-button in the top left corner of the Gamewindow. Clicking this button hides the Gamewindow, while still leaving the game running. If you do this, you'll notice the "NEW GAME"-button has changed into a "RESUME GAME"-button. Clicking on that button brings you back to the Gamewindow. Closing the program should always be done from the Main Menu.

Image 4: The Gamewindow

The first thing the player probably notices when opening the Gamewindow, is the graphical representation of the map. The player starts in the top left corner of the map, but the map is larger than this view. To see the rest of the map, you can either scroll the view with the scrollbars, or use the arrowkeys on the keyboard.

The Gamewindow also has an interface for building buildings, and another for giving commands to settlers. The buildings are located on the left side of the mapview, and the commands are above it.

So far, the game does not seem to be doing much. To change this, tell a settler to go chop down some trees. To do this, first, left-click on the settler in the house. Then, left-click on the "GATHER WOOD"-icon in the "Commands"-interface. You will know it worked, when the settler exits the house, goes to the closest tree, spends some time chopping it and then heads to the warehouse to drop of wood. This settler will continue chopping wood until given another command.

To make a new building, for example a stonecutter, left-click on the "STONECUTTER"-icon in the "Buildings"-interface and then left-click on any terrain next to a stone, for example the one under the house, or the one in the top left corner. This will create a "constructionsite" in the place you clicked.

If you at any point decide that you don't want to place a building or give a command, simply right-click on any terrain. This takes you out of build- or commandmode.

To actually build the stonecutter, you need to tell a settler to build it. There is still another settler in the house. Click on the settler, and then click on the "BUILD"-icon in the "Commands". The settler will now get wood from the warehouse, walk to the constructionsite and start building. When the settler runs out of wood, he gets more from the warehouse. After the stonecutter is

finished, the settler will find the closest house to hang out in while waiting for new constructionsites to pop up. The settler will continue doing this until given some other command.

There is also Bob and he is jealous of your empire. Good thing he lives in an island far from you and he can’t get to you. Or can he? When Bob attacks, you must have your settlers prepared. To fight Bob just give your settlers the combat task. If you have swords in the warehouse, the settlers will automatically arm themselves while in combat task.

Finally, to exit the program, click on the "Main Menu"-button in the Gamewindow, and then on the "QUIT"-button in the Main Menu.

4. Testing

Testing of the GUI was done manually, utilizing debug-messages printed in the terminal, letting the game run for an extended amount of time, moving settlers around to random coordinates, using timers to create new settlers and buildings in random coordinates and destroying these, and of course, playtesting and trying to break either the GUI, the game, or both. One of the major revelations during these tests was that, for some reason, the coordinates for the terrain had been flipped somewhere in the code. Other examples of things found through testing include noticing that settlers are fond of climbing trees after building things, settlers trying to chop wood when they should have been cutting stone, buildings disappearing randomly, non-existent settlers staying behind as ghosts when scrolling the view (still haven't found a solution for this one), roads changing shape, resource buildings not containing resources and settlers picking up imaginary things from warehouses and trying to build with them. Sometimes the program would just crash with a segmentation fault, usually caused by a building accidentally disappearing.

Memory management was tested using Valgrind. It seems Valgrind does not play nice with Qt, but according to frantic searching of the internet and asking on Slack, this appears to be normal. Disabling the GUI altogether, running the game without any graphics, no memory leaks were detected.

5. Work log

When dividing up the work, attention was given to experience in, or, more importantly, interest shown by group members in specific subjects, like GUI-development, AI and implementing algorithms.

Based on previous work during the second course in Python (Basic Course in Programming Y2), Arthur was interested in making the graphics and GUI for the game, so he focused mostly on that.

The group had roughly four weeks to make the game, after submitting the plan for the project. Not much had been done before that. Initially, in the plan, the goal for the first week was to create a basis for the GUI, a working map for the game, and a basis for the core of the game. Goals for the second week included things like functioning settlers and buildings, further development of the GUI and implementing the pathfinding Djikstra-algorithm. During the third week it was planned that the group would work on combat mechanics, implementing a “TaskHandler”-AI and doing finishing touches on the GUI and the game in general. Since we assumed the deadline would be in the beginning of the fourth week, we did not plan that far ahead.

Work log, per week, rough estimate:

	Week 1	Week 2	Week 3	Week 4
Joonas Lehto	Planned structure for game core classes, helped creating map and terrain classes, project plan. ~10hours	Worked on path finding algorithm and writing up classes and functions ~15hours	Worked out logic how AI should handle different tasks, implementing functionalities ~20hours	Implementing missing functionalities and fixing major bugs in GUI, ~30hours
Joonas Malinen	Created map and terrain classes and project plan. ~10 hours	Worked on Game function, so that we could create the map ~15hours	Implemented functionalities to game so that it runs ~30hours	Finalizing missing functions and other finishing touches ~30hours
Tarmo Nieminen	Revising the classes, inheritance and other basic things in C++, also learning how to use git. Creating small programs and thinking about the basic class structure of the game. ~15ish hours.	Figuring out what the basic classes are supposed to be like and creating them. ~10ish hours.	Changing previously added classes to better suit the game e.g. changing variable types. Familiarizing myself with the other functions that run the game. ~10ish hours.	Fixing some minor problems with the game. ~5 hours.
Arthur Löfstedt	Teaching myself Qt by reading documentation, watching tutorials and creating small test-programs. Trying to re-create my project from the Python-course in C++. ~10ish hours.	Starting to work on the basis of the GUI, learning window inheritance the hard way. Learning git. First commit. More tutorials, not counting these as working hours though. ~10 hours.	Serious work on GUI and graphics, searching for, creating and modifying images, adding interactivity. ~25 hours.	Adding more interactivity to GUI, polishing, bugfixing, writing documentation. ~35 hours.